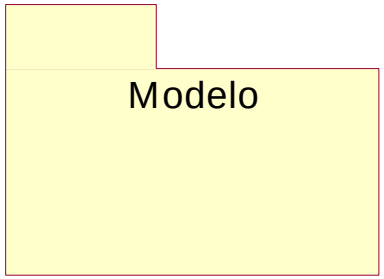
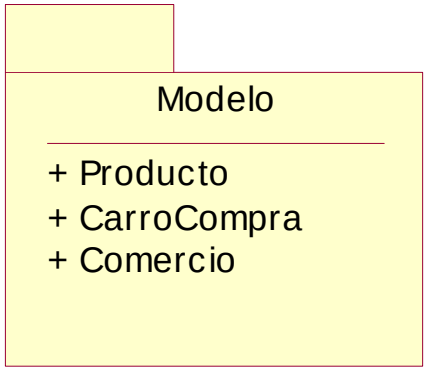
Paquetes

- Elemento organizativo
- Puede contener elementos de cualquier tipo.
- Un elemento es exclusivo a un paquete.
- Posibilidad de anidar paquetes.



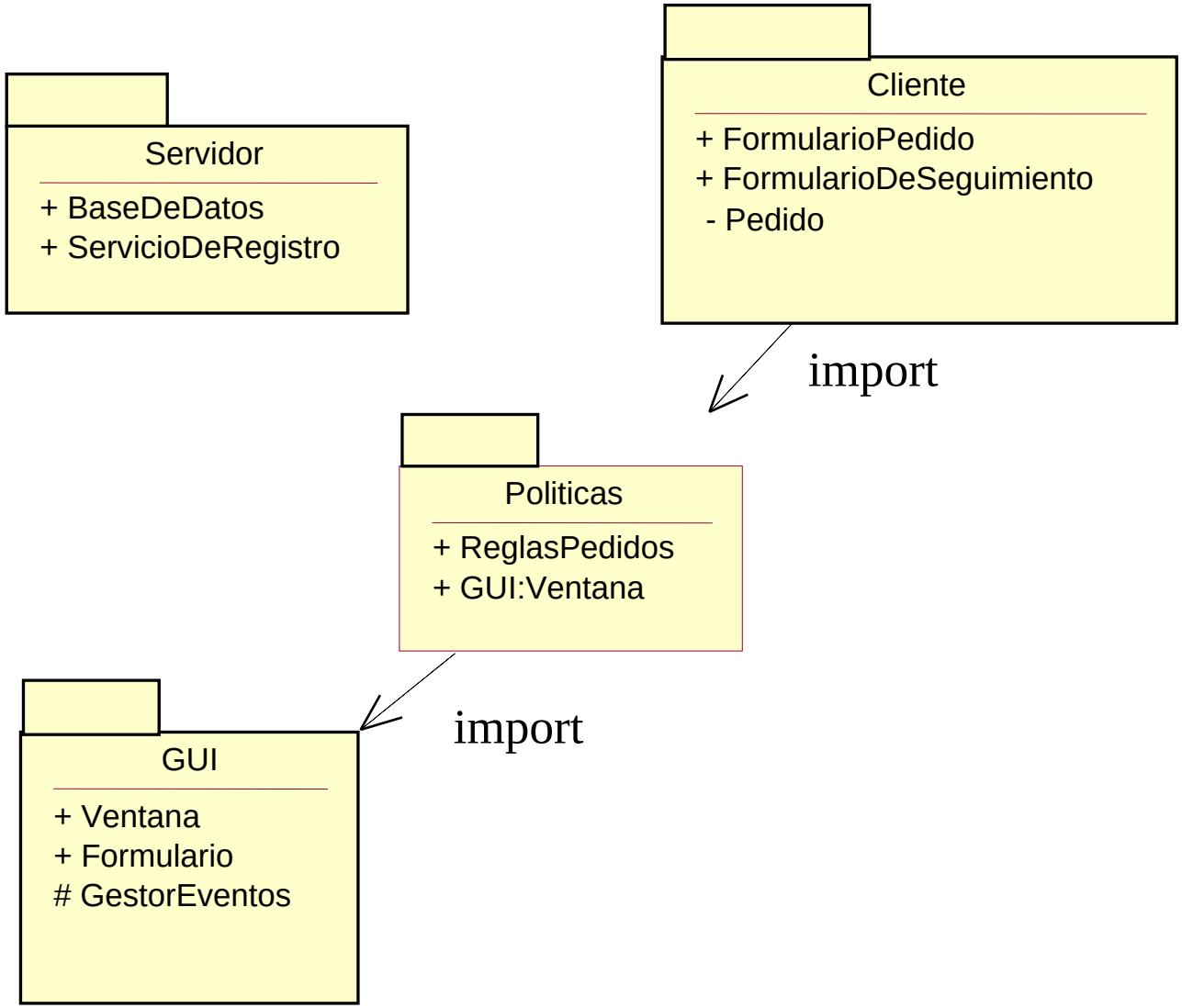


Paquetes

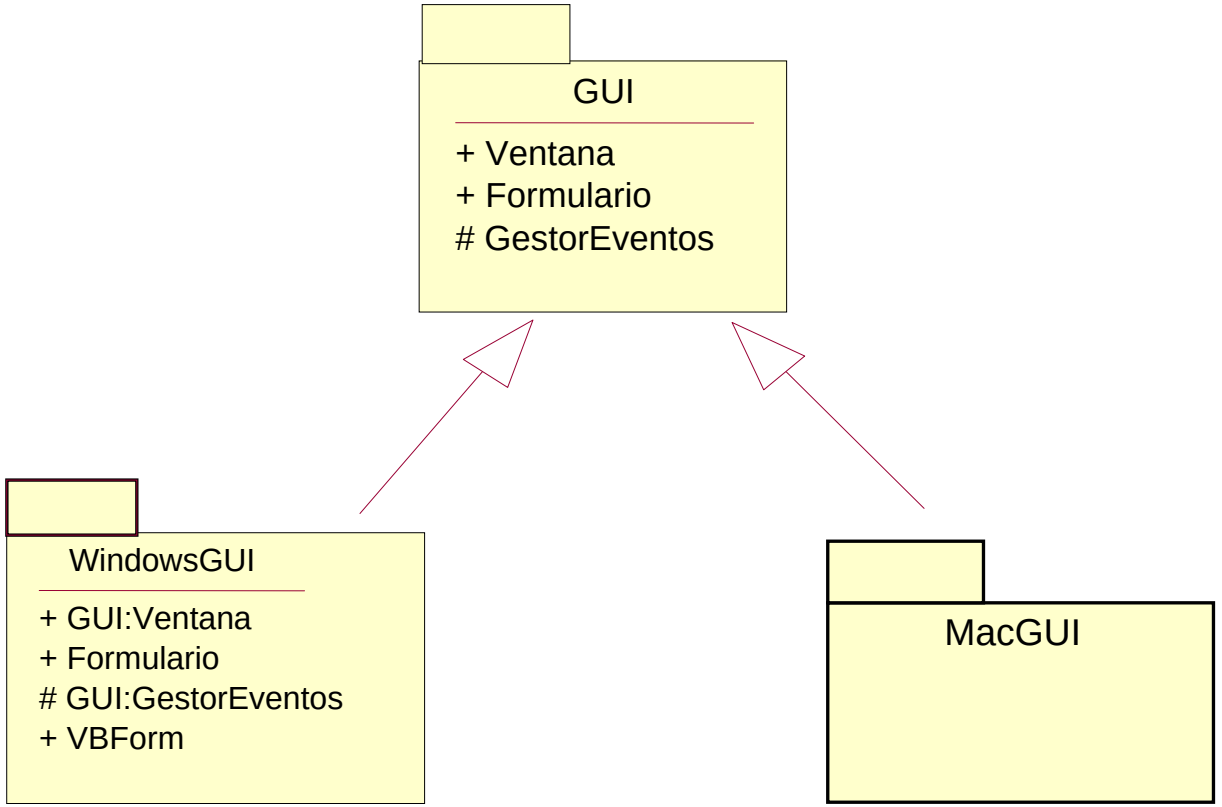
- Un paquete bien estructurado debe:
  - ser cohesivo
  - estar poco acoplado
  - poco anidamientos
  - conjunto equilibrado de elementos

Importación/Exportación en paquetes

- *Los paquetes permiten controlar la complejidad del manejo de un gran número de abstracciones, controlando los accesos mediante la importación.*
- La parte pública de un paquete son sus exportaciones.
- Las partes públicas son visibles en los paquetes que importan al paquete contenedor.
- La importación no es transitiva.
- Los paquetes anidados pueden ver todo lo que ven los paquetes que los contienen.


Servidor
+ BaseDeDatos
+ ServicioDeRegistro
Cliente
+ FormularioPedido
+ FormularioDeSeguimiento
- Pedido
import
Politicas
+ ReglasPedidos
+ GUI:Ventana
import
GUI
+ Ventana
+ Formulario
# GestorEventos

Generalización de Paquetes



Uso de los paquetes

- Agrupar elementos, normalmente del mismo tipo, para manejarlos en conjunto.

  Paquete *"Clases e interfaces del modelo"*

  Paquete *"Interfaces de usuario"*

  Paquete *"Servicios base de datos"*

- Un modelo es un paquete que incluye todos los elementos que constituyen una particular vista del sistema modelado.

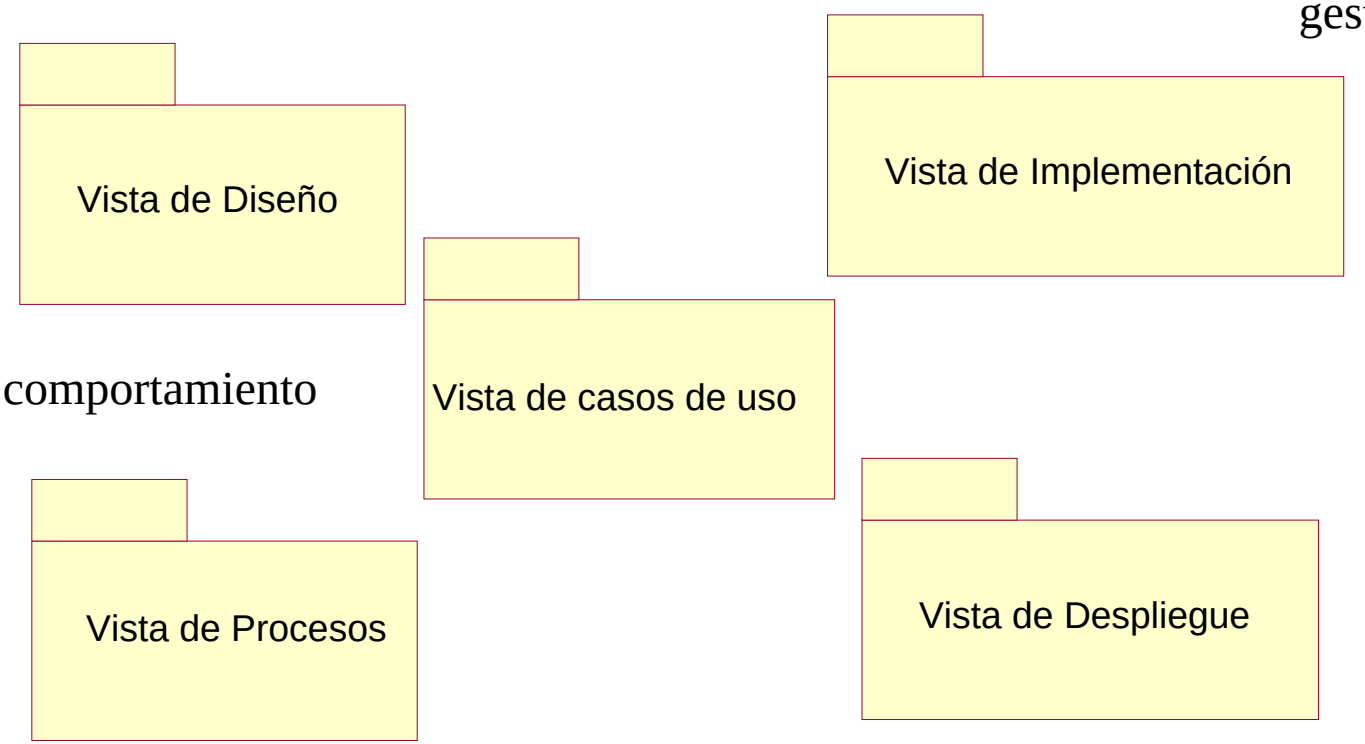
Sistema, modelo, vista, diagrama

- Un sistema es aquello que se está desarrollando y para lo que se crean modelos.
- Un subsistema es una parte de un sistema.
- Un modelo es una abstracción de un sistema que ayuda a comprenderlo.
- Una vista es una proyección de la estructura y organización de un modelo del sistema, centrada en algún aspecto.
- Un diagrama es una representación de un conjunto de elementos de un modelo.

Vistas UML

vocabulario
funcionalidad

ensamblado
gestión conf.



comportamiento

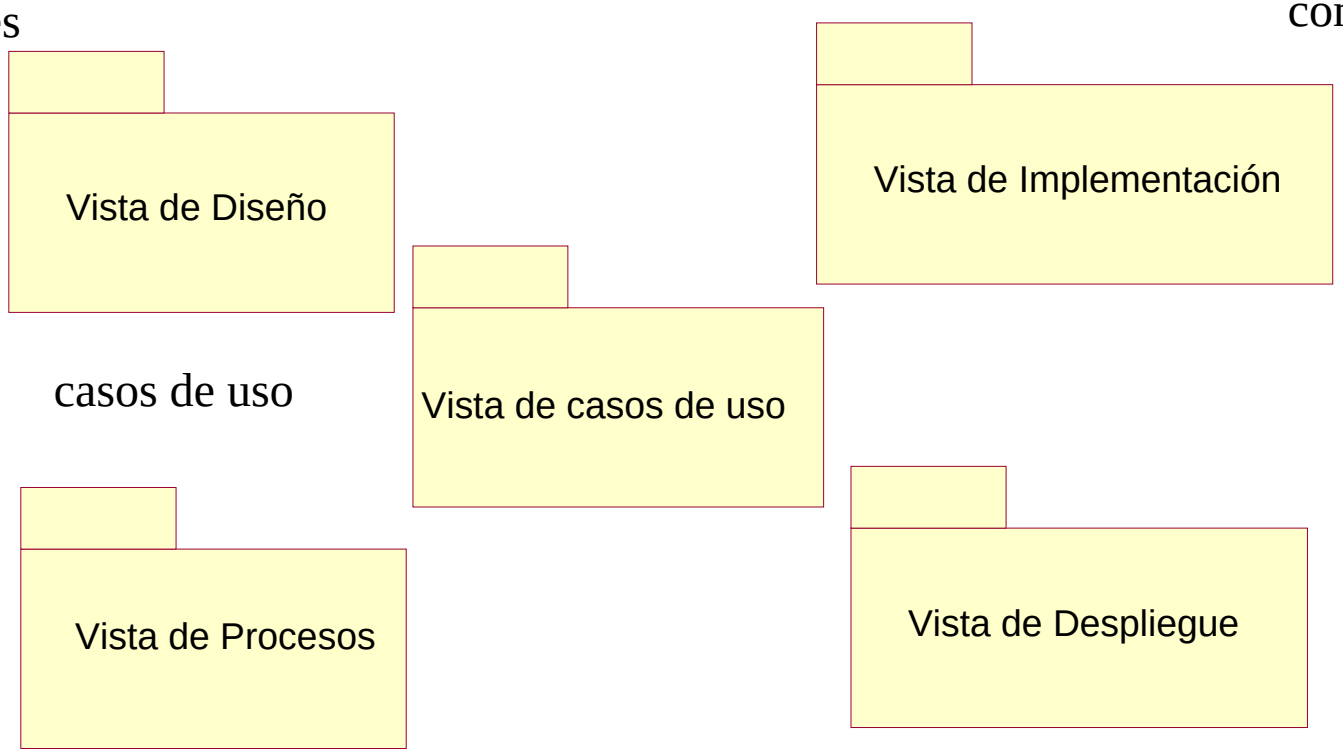
topología
entrega
distribución
instalación

Funcionamiento
escalabilidad
rendimiento

Vistas UML

clases
interfaces
colaboraciones

componentes



casos de uso

clases activas

nodos

Vistas UML

Diagramas de clase
Diagramas de interacción
Diagramas de estado

Vista de Diseño

Diagramas de componentes
Diagrama de interacción
Diagramas de estado

Vista de Implementación

Diagramas de casos de uso

Vista de casos de uso

Vista de Procesos

Vista de Despliegue

Diagramas de clase
Diagramas de interacción
Diagramas de estado

Diagramas de despliegue
Diagrama de interacción
Diagramas de estado

Diagrama de Objetos

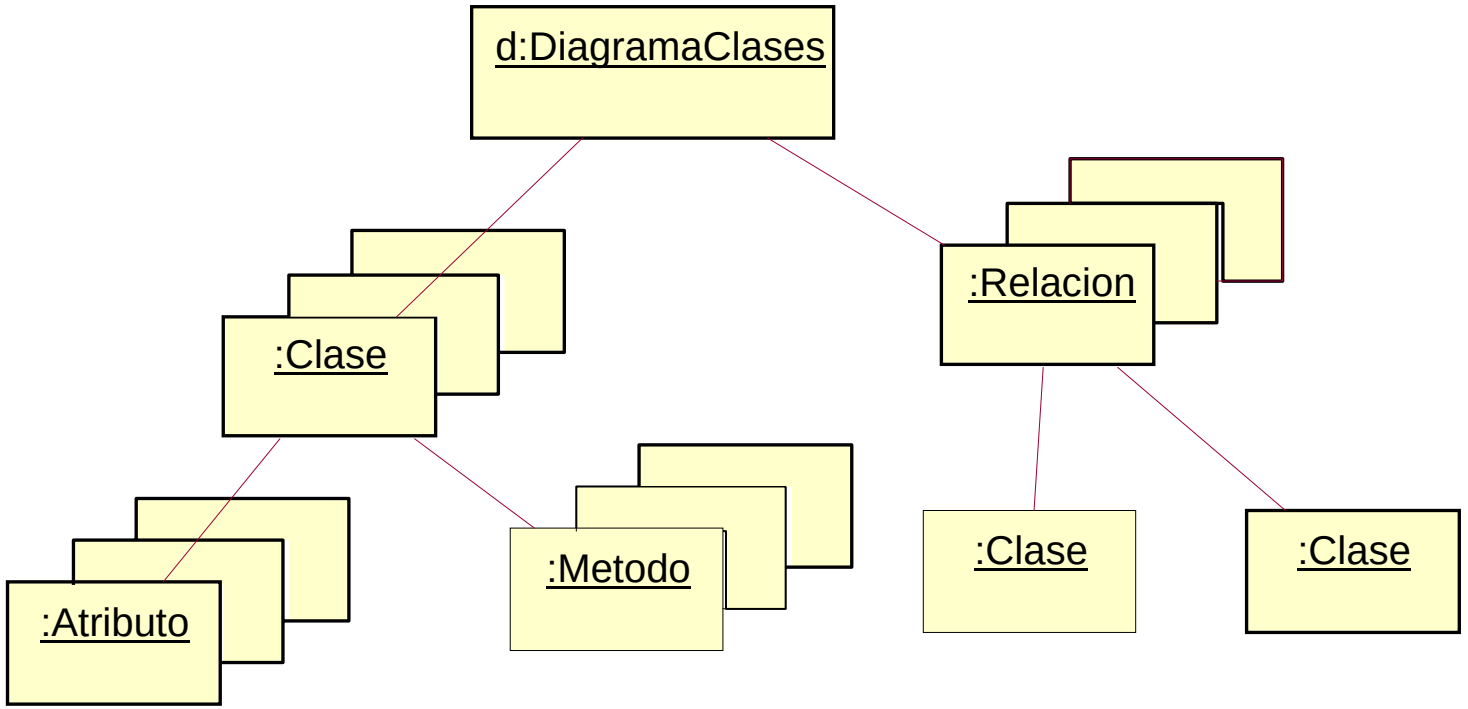
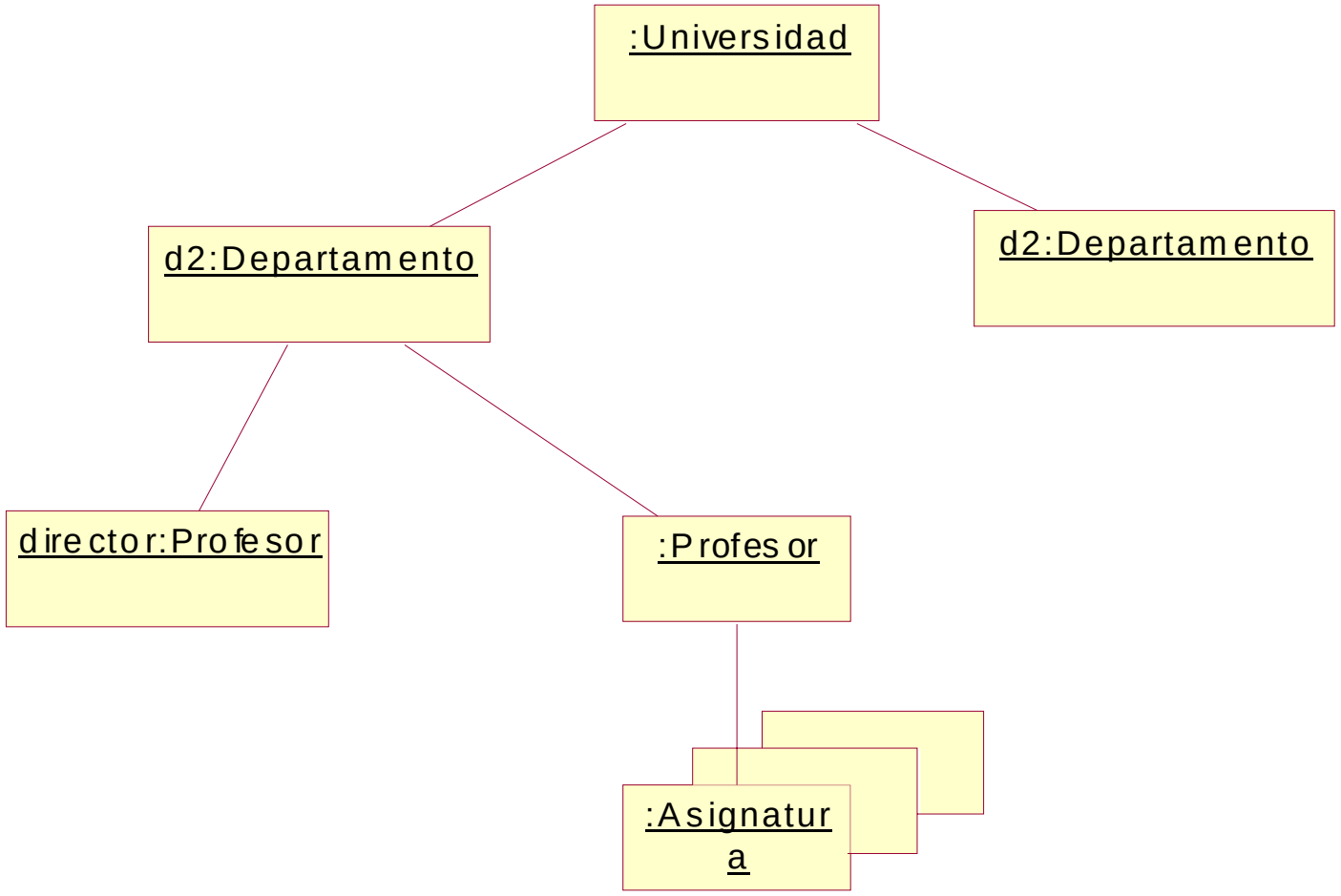
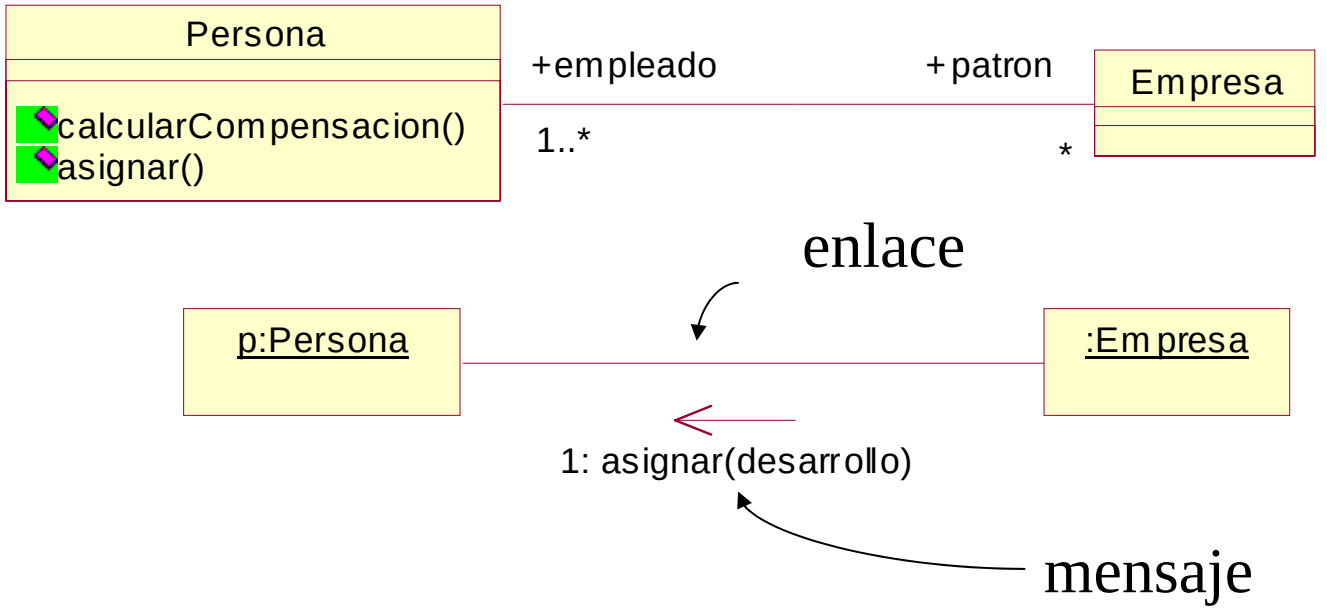


Diagrama de Objetos



Enlaces y Asociaciones

- Un enlace es :
  - una conexión semántica entre objetos.
  - una instancia de una asociación.
  - un camino por el cual enviar un mensaje



Interacciones y Mensajes

- **Interacción**: comportamiento que comprende un conjunto de mensajes intercambiados entre un conjunto de objetos dentro de un contexto para lograr un propósito.
- **Mensaje**: especificación de una comunicación entre objetos que transmite información, con la expectativa de desencadenar una actividad.

Modelado del comportamiento

- Se describe cómo los objetos colaboran entre sí para realizar cierta actividad.
- Se expresa mediante los **diagramas de interacción**:
  - Diagramas de Secuencia y Diagramas de Colaboración (de Comunicación en UML 2.0).
- También se describe las máquinas de estado que caracterizan los objetos
  - **Diagramas de estado**
  - **Diagramas de actividades**

Diagramas de Interacción

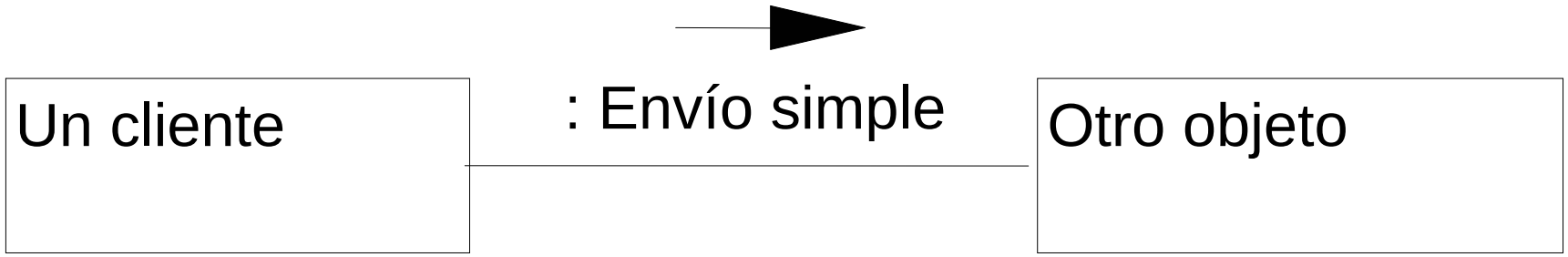
- Describen una interacción.
- Dos tipos: Diagramas de Secuencia y Colaboración
- Diagramas de **Secuencia**:
  - Destacan la ordenación temporal de los mensajes
- Diagramas de **Colaboración**:
  - Destacan la organización estructural de los objetos participantes.
- Equivalencia semántica

Diagramas de Secuencia

- Representan escenarios.
- Incluye:
  - **Objetos** y su **línea de tiempo**
  - **Mensajes** : argumentos y **self-delegation**
  - Información de control: **condiciones y marcas de iteración**
  - Indicar el objeto devuelto por el mensaje**: return**
    (añadirlos sólo cuando ayuden a clarificar la interacción)
  - Especificar procesos concurrentes: **focos de control**

Sincronización

- Envío de mensaje simple:
  - Un solo hilo de ejecución
  - El objeto activo pasa el control al objeto pasivo



Sincronización

- Envío de mensaje síncrono:
  - Sólo se desencadena la operación cuando el servidor acepta el mensaje
  - El cliente se bloquea hasta que el servidor lo acepta o rechaza

Sincronización

- Envío de mensaje asíncrono:
  - Un mensaje asíncrono no interrumpe la ejecución del cliente
  - El cliente envía el mensaje sin saber cuándo ni si se llevará a cabo

Diagrama de secuencia

- Un objeto puede enviarse a sí mismo un mensaje:

Puede representar también la entrada por parte del objeto en cierta actividad de más bajo nivel

Diagrama de secuencia

- Gráficamente también se puede indicar cuándo el mensaje es para crear el objeto (va dirigido al rectángulo del objeto o etiquetado con *new*) o para destruirlo (va dirigido a la línea del objeto pero el final de la flecha es una cruz)

Diagrama de secuencia

- Normalmente no es necesario indicar el retorno del control, cuando el envío es síncrono:

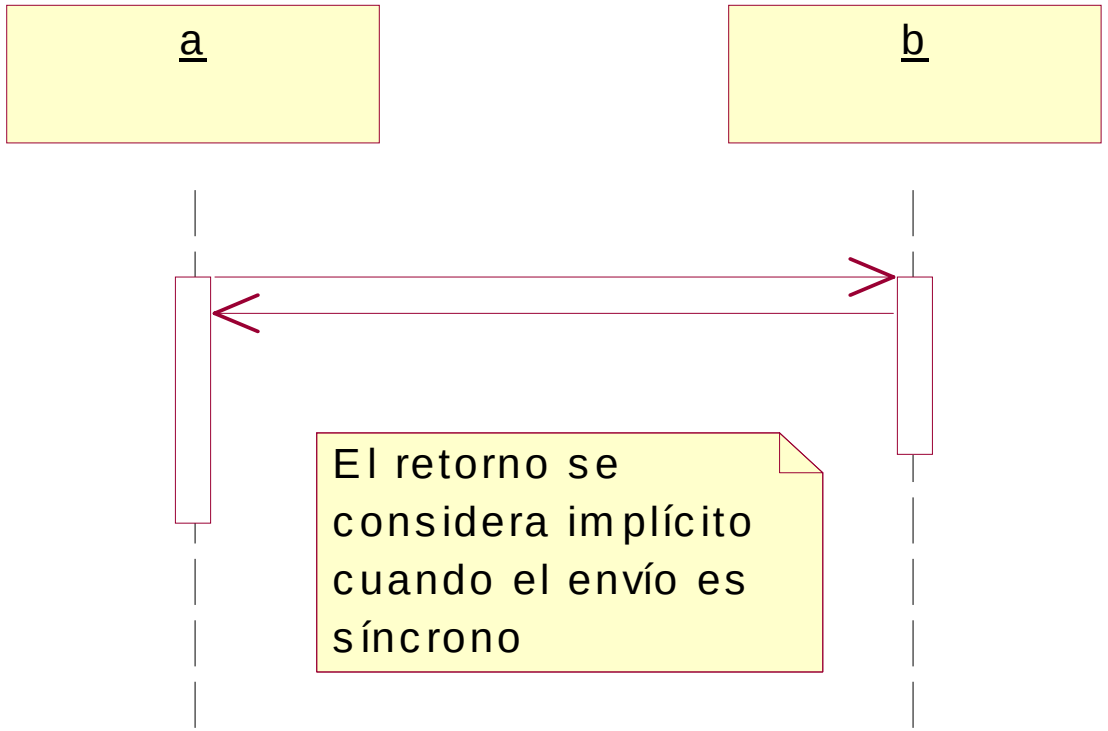
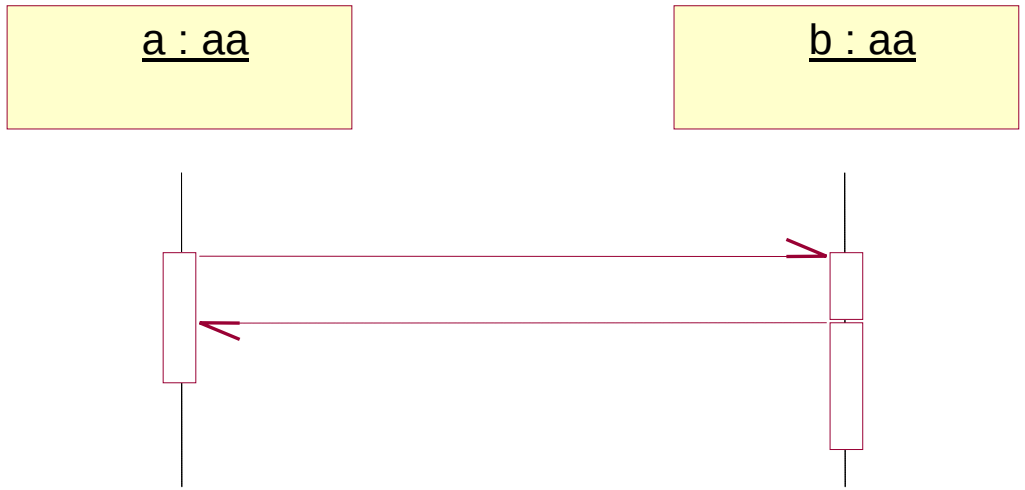
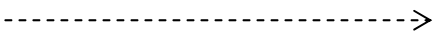


Diagrama de secuencia

- En el caso asíncrono el retorno, si existe, se debe representar:



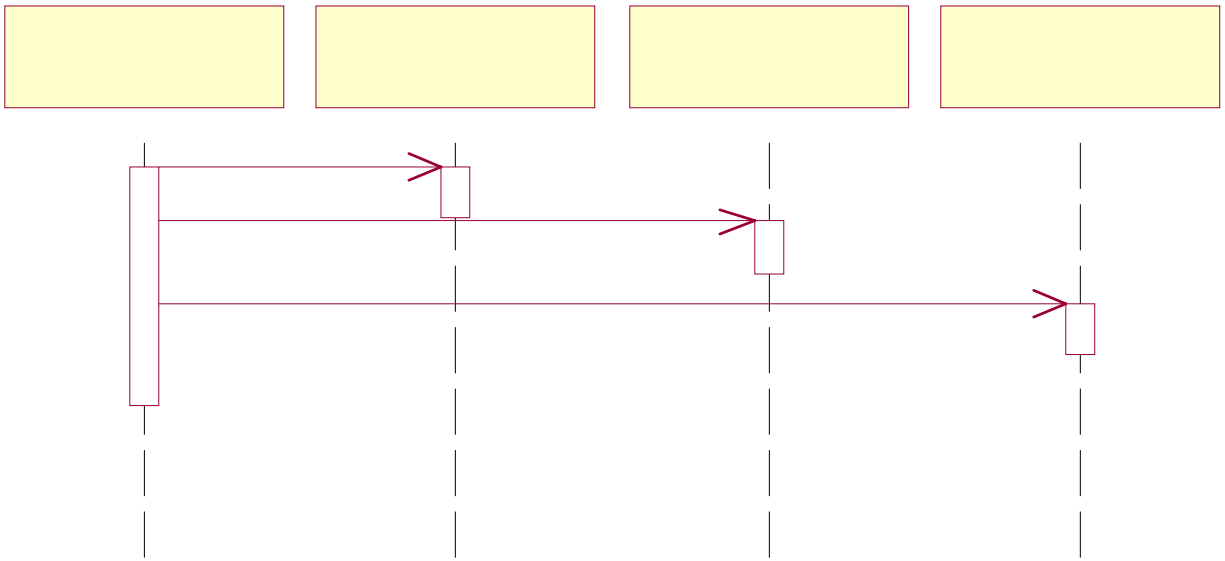
Media punta de flecha indica mensaje asíncrono (desde UML 1.3; en UML 1.5 ya no se distingue, aunque se acepta)



La flecha discontinua indica retorno del control

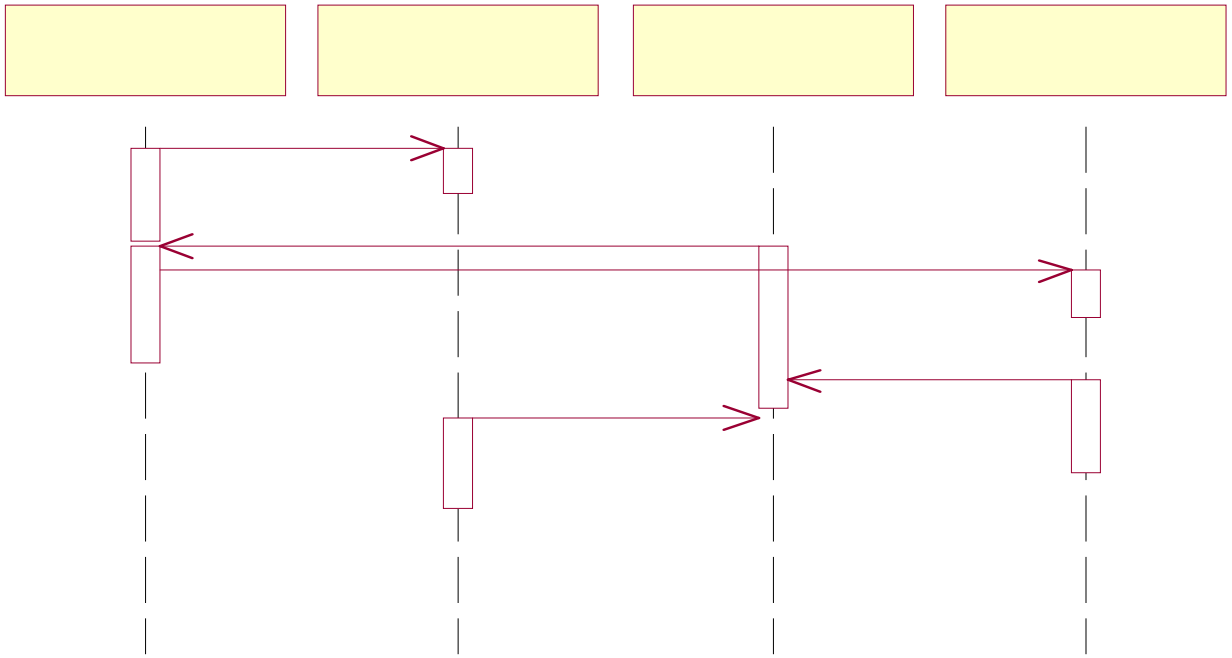
Tipos de Control

- El Diagrama de Secuencia refleja de manera indirecta las opciones de control
- Un control centralizado tiene una forma como esta:



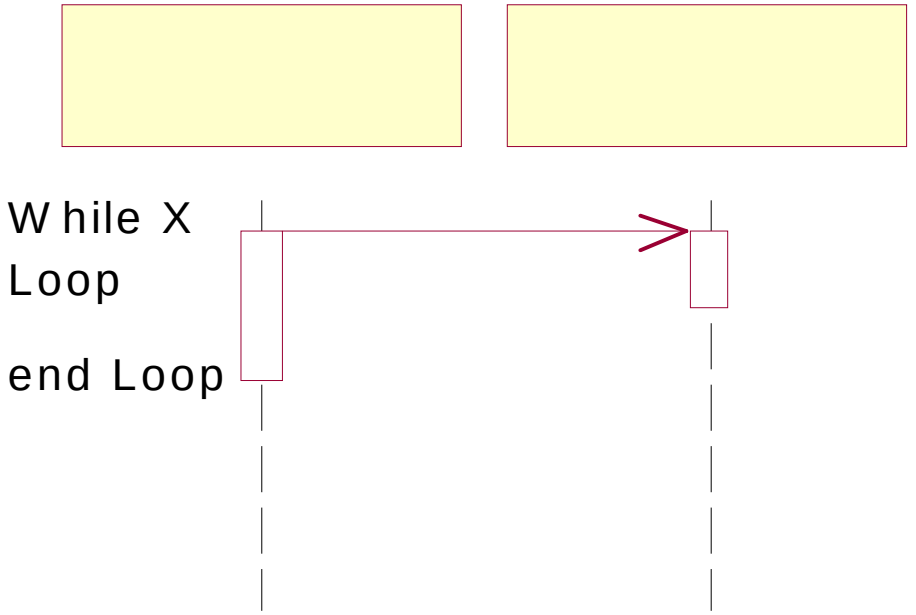
Tipos de control

- Un control descentralizado tiene una forma como ésta:



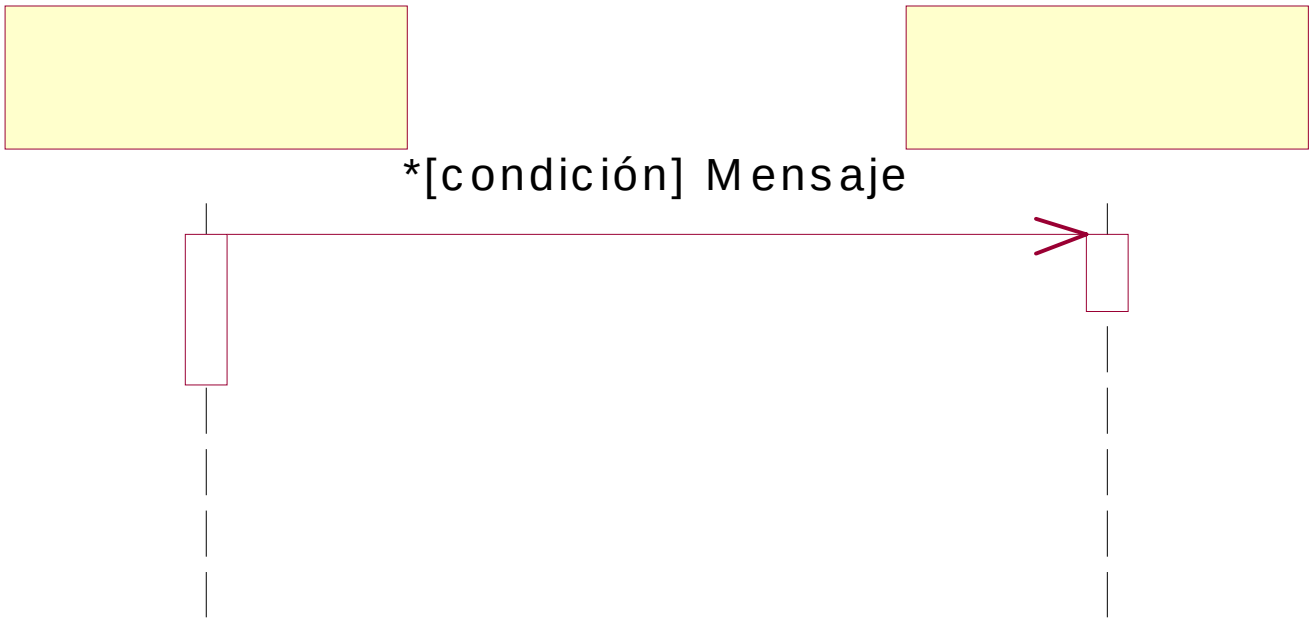
Estructuras de control

- Podemos representar iteraciones en el envío de mensajes mientras, p.e., se cumpla una condición:



Estructuras de control

- La iteración puede expresarse también como parte del mensaje:



Estructuras de control

- Las bifurcaciones condicionales pueden representarse de esta forma:

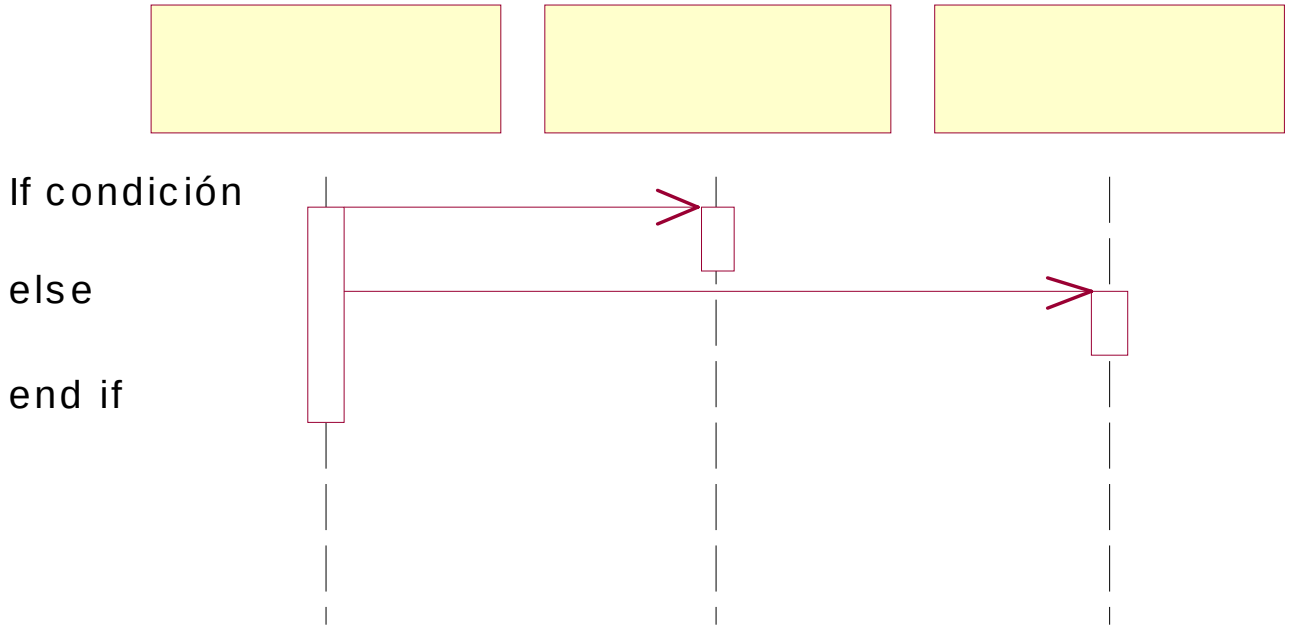
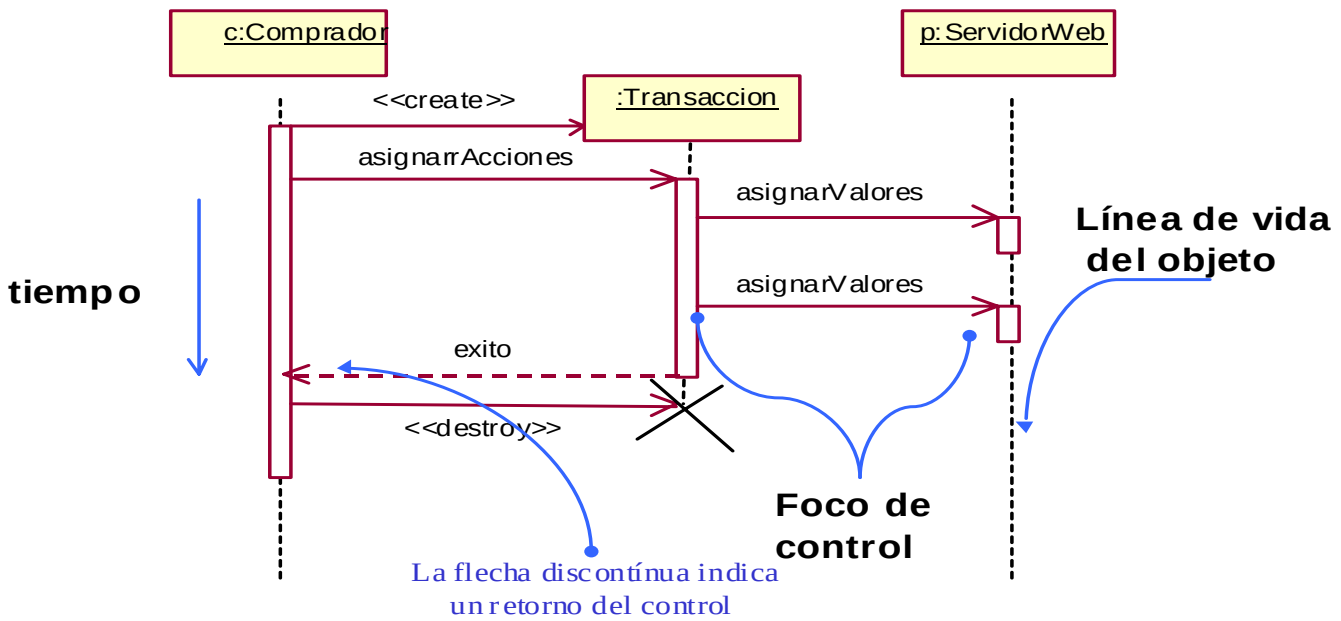


Diagrama de secuencia



Foco de control:Muestra el periodo de tiempo que un objeto está realizando una acción (períodos de actividad del objeto). La parte superior indica cuándo se inicia la acción y la inferior cuándo se termina (puede estar marcada con un mensaje de respuesta -return-, indicando que se devuelve el control al objeto destino -se indica con flecha discontinua-)

Diagrama de Colaboración

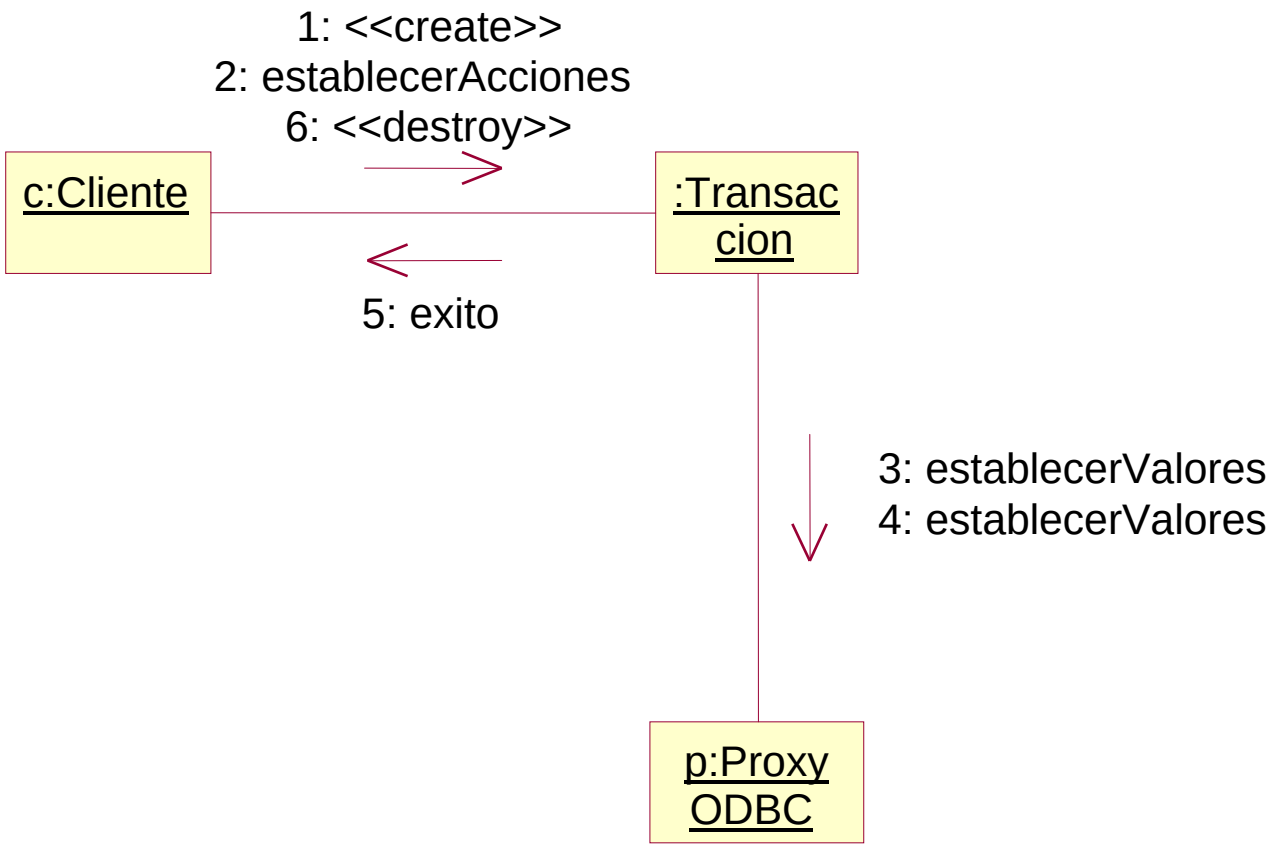


Diagrama de Secuencia

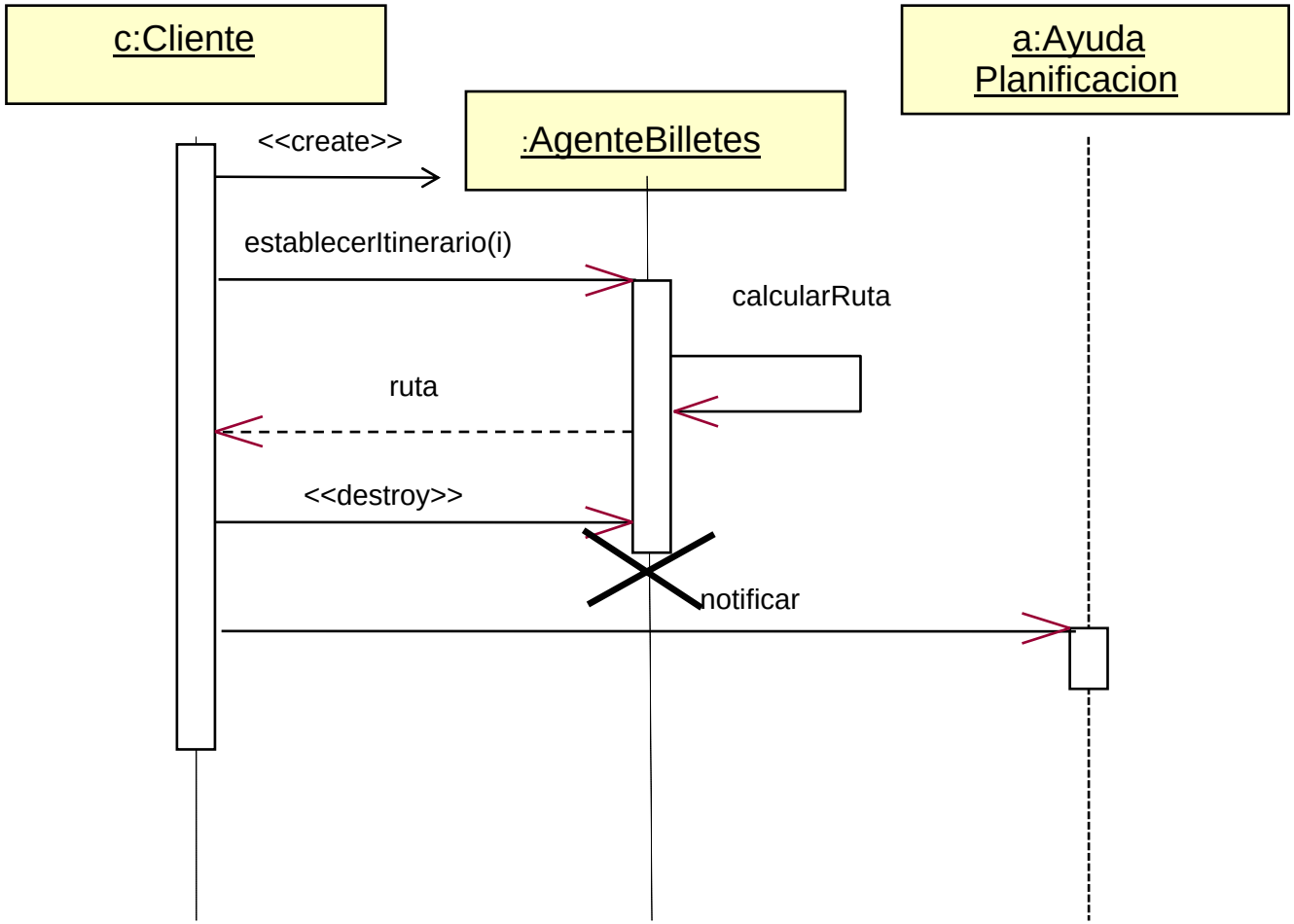
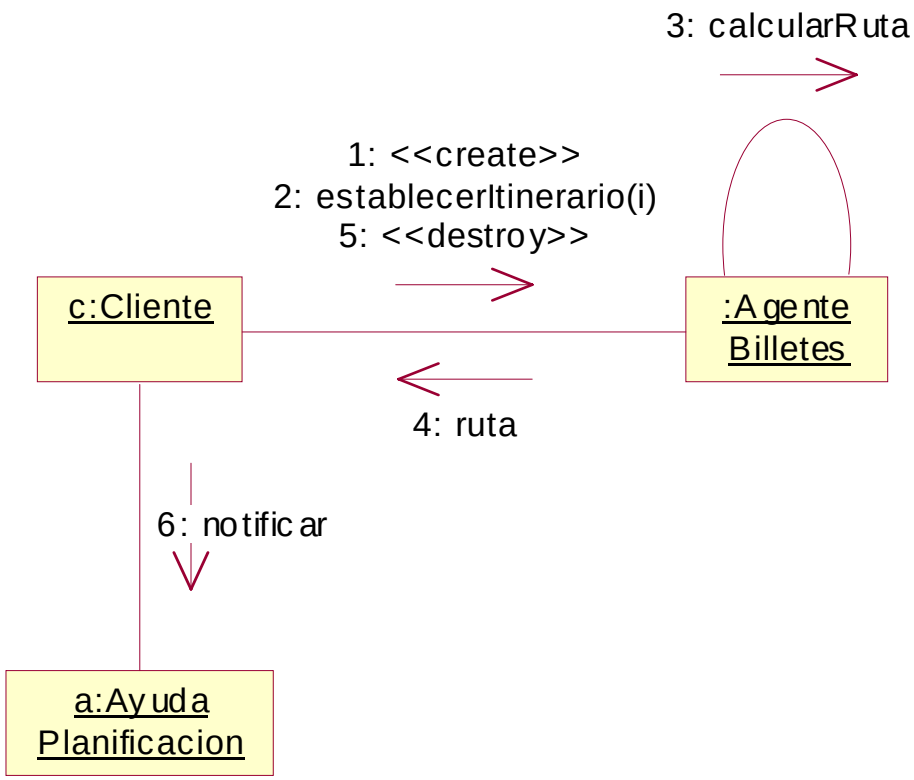
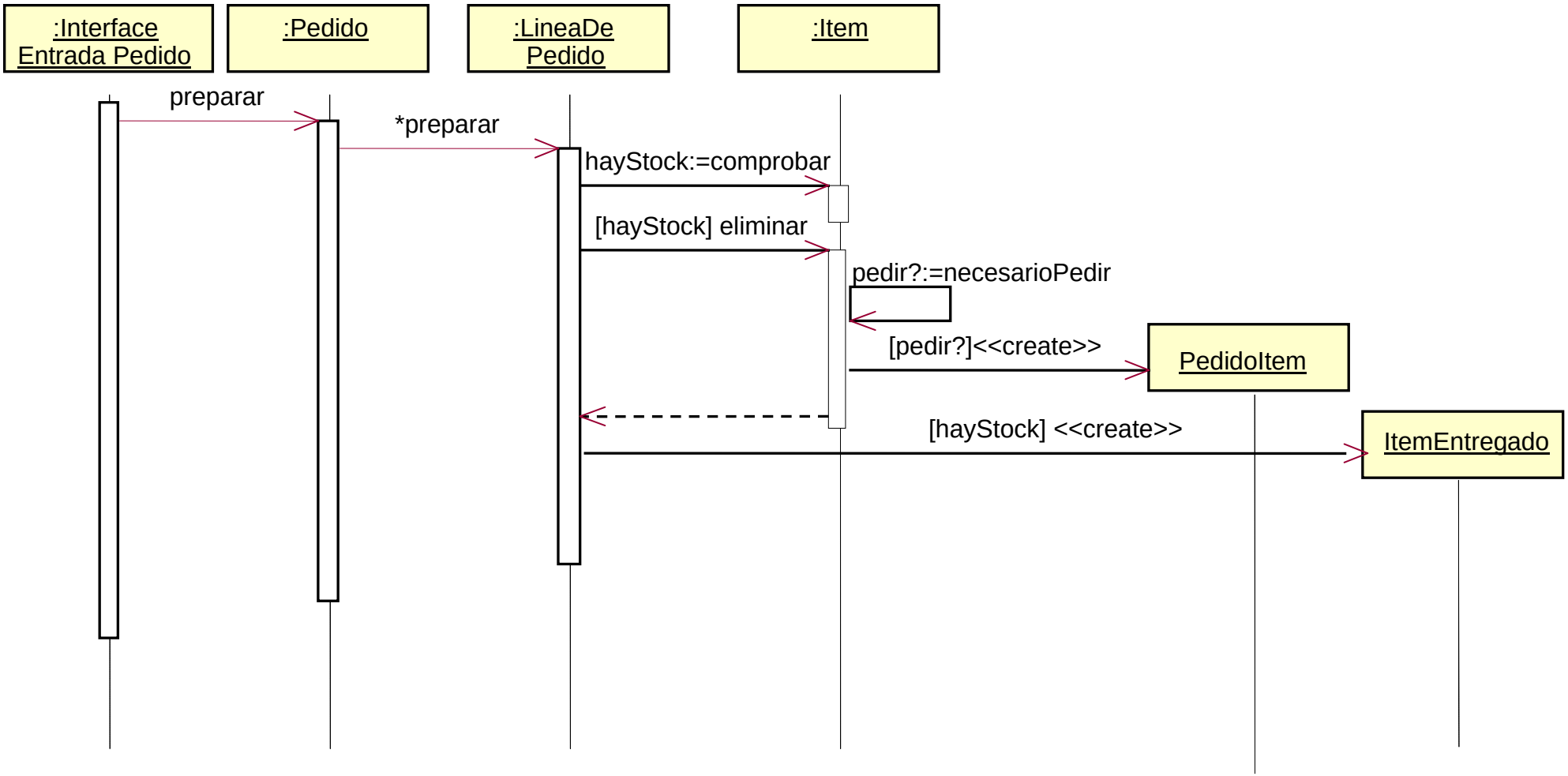


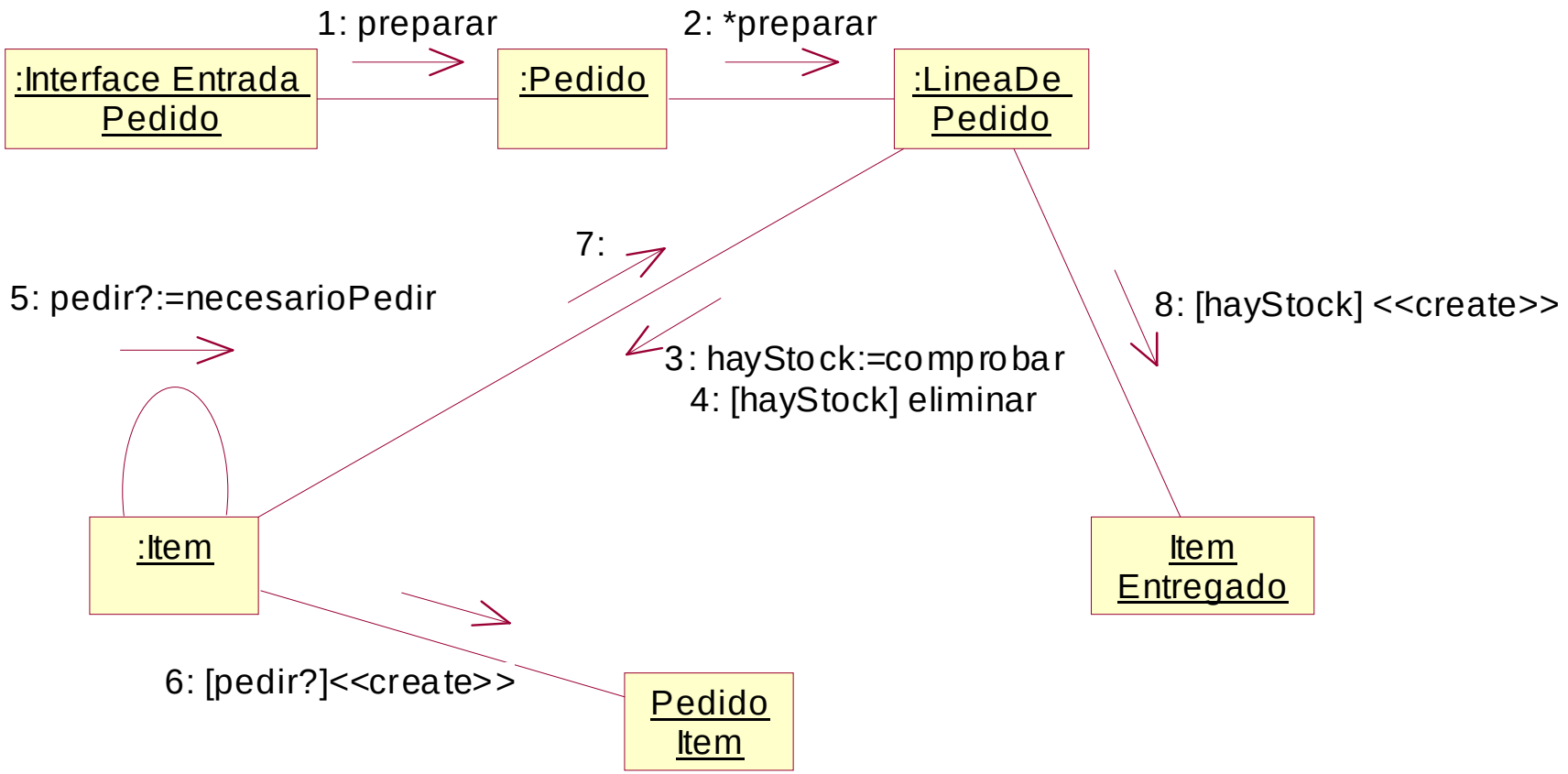
Diagrama de Colaboración



Diagramas de Secuencia



Diagramas de Colaboración



Uso de los diagramas de interacción

- Modelado del aspecto dinámico.
- Modelado del flujo de control que caracteriza el comportamiento de un sistema:
  - casos de uso
  - colaboraciones
  - patrones
  - *frameworks*
  - operaciones

Diagrama de Secuencia (Caso de uso)

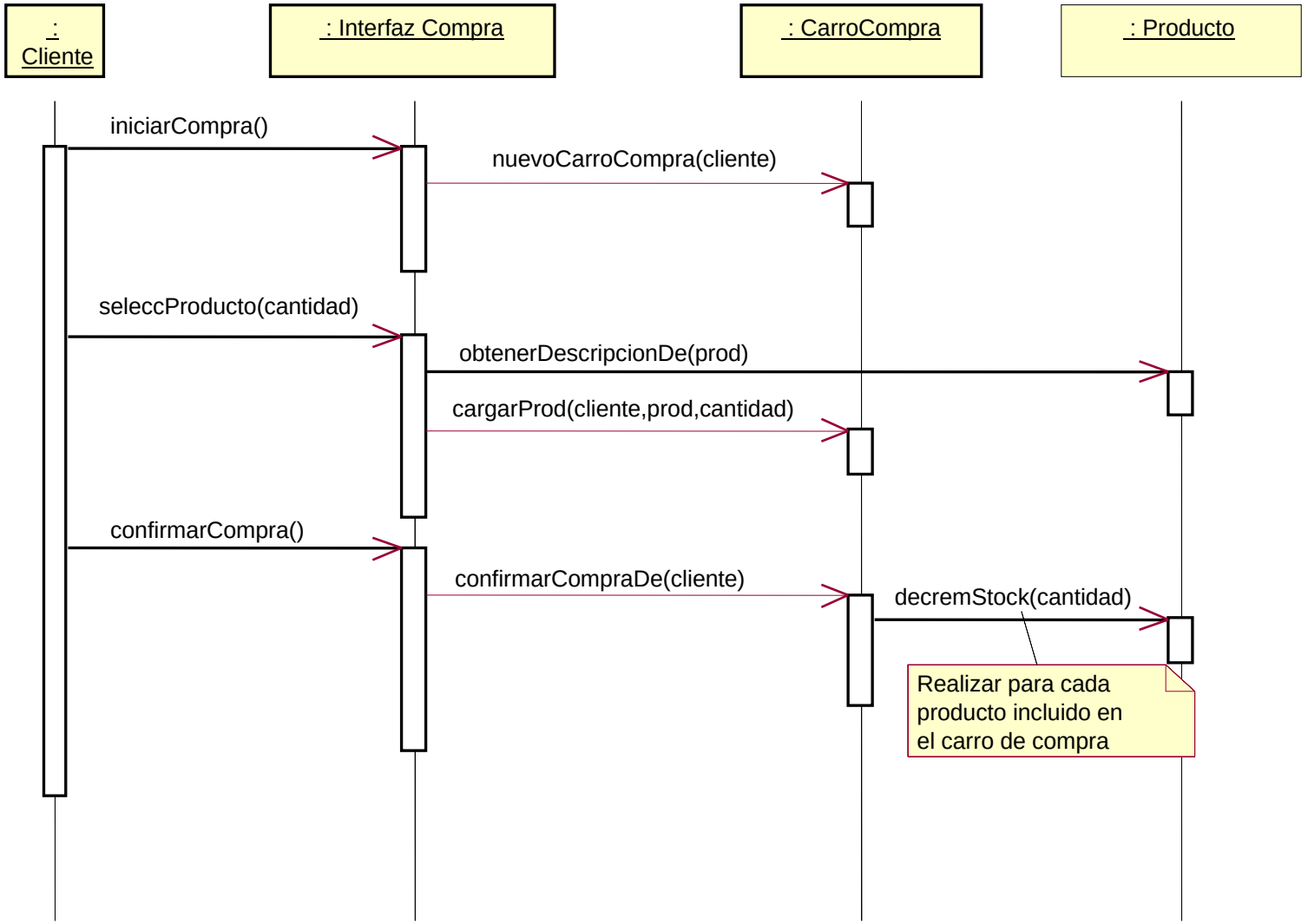


Diagrama de Colaboración Caso de Uso)

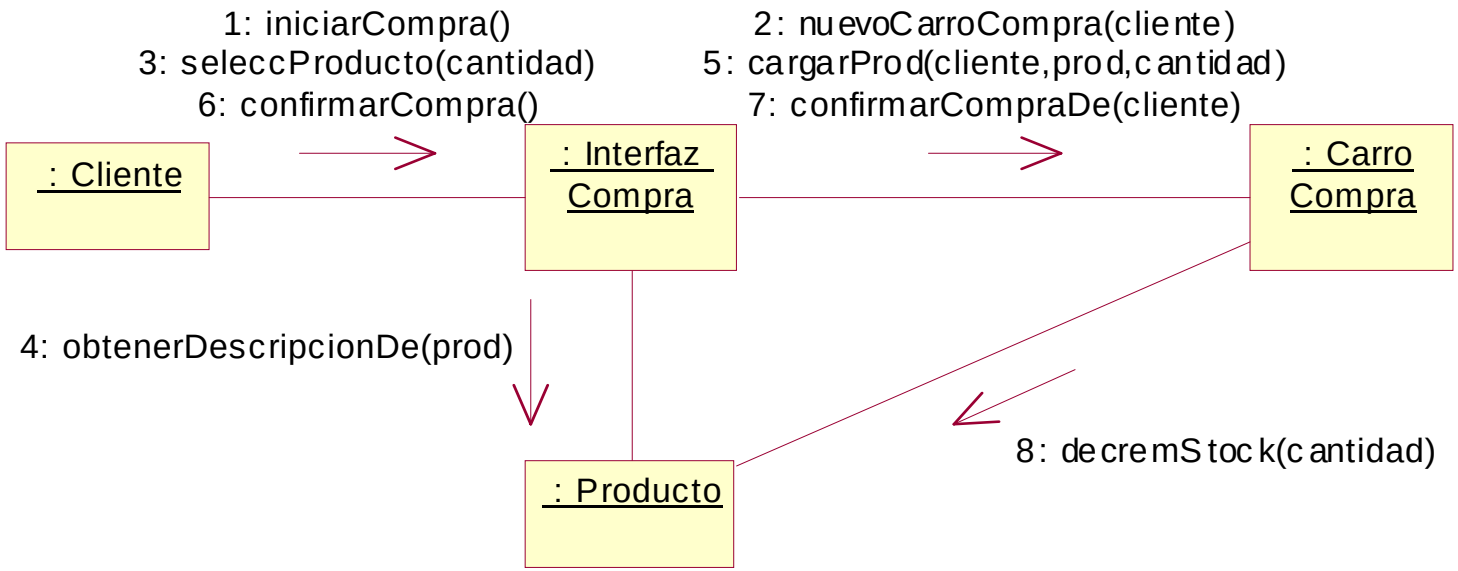
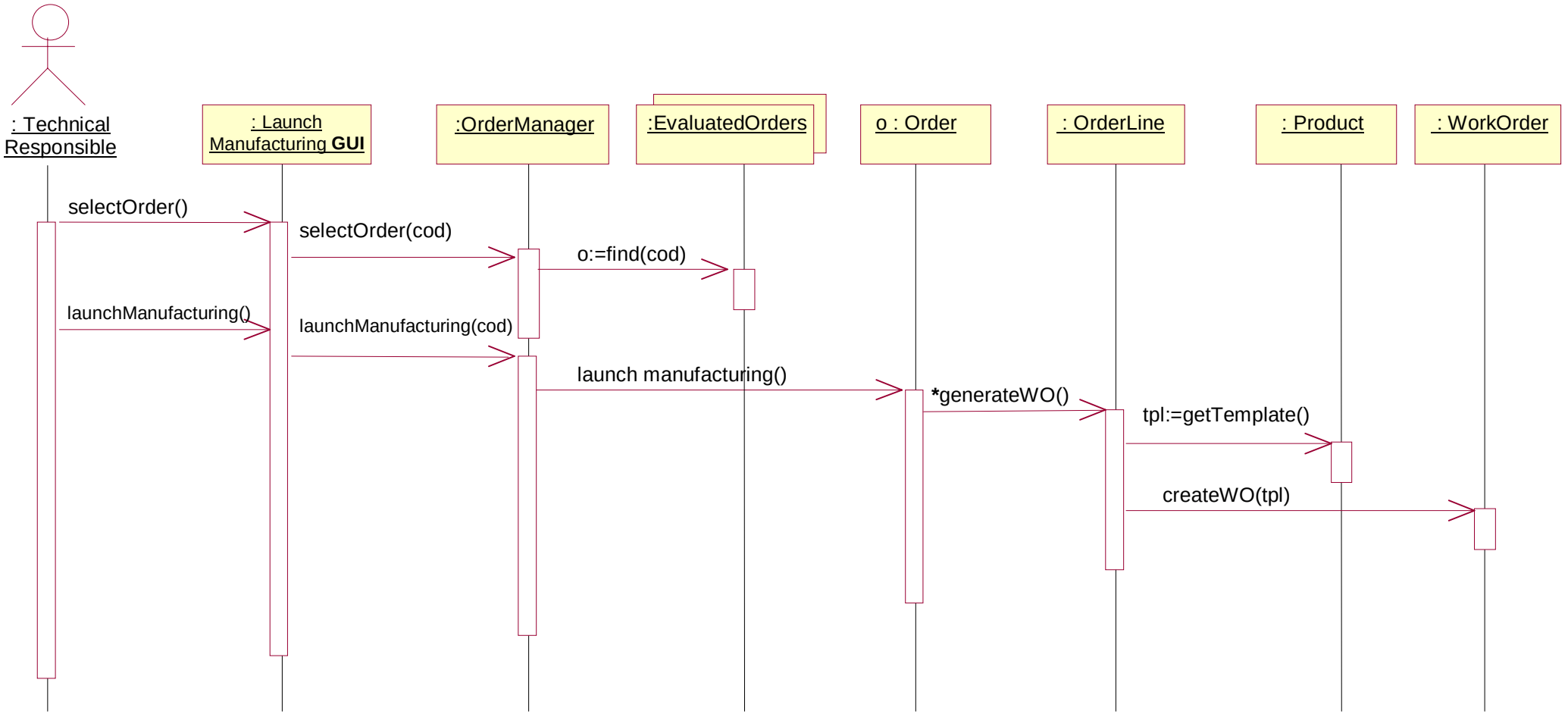
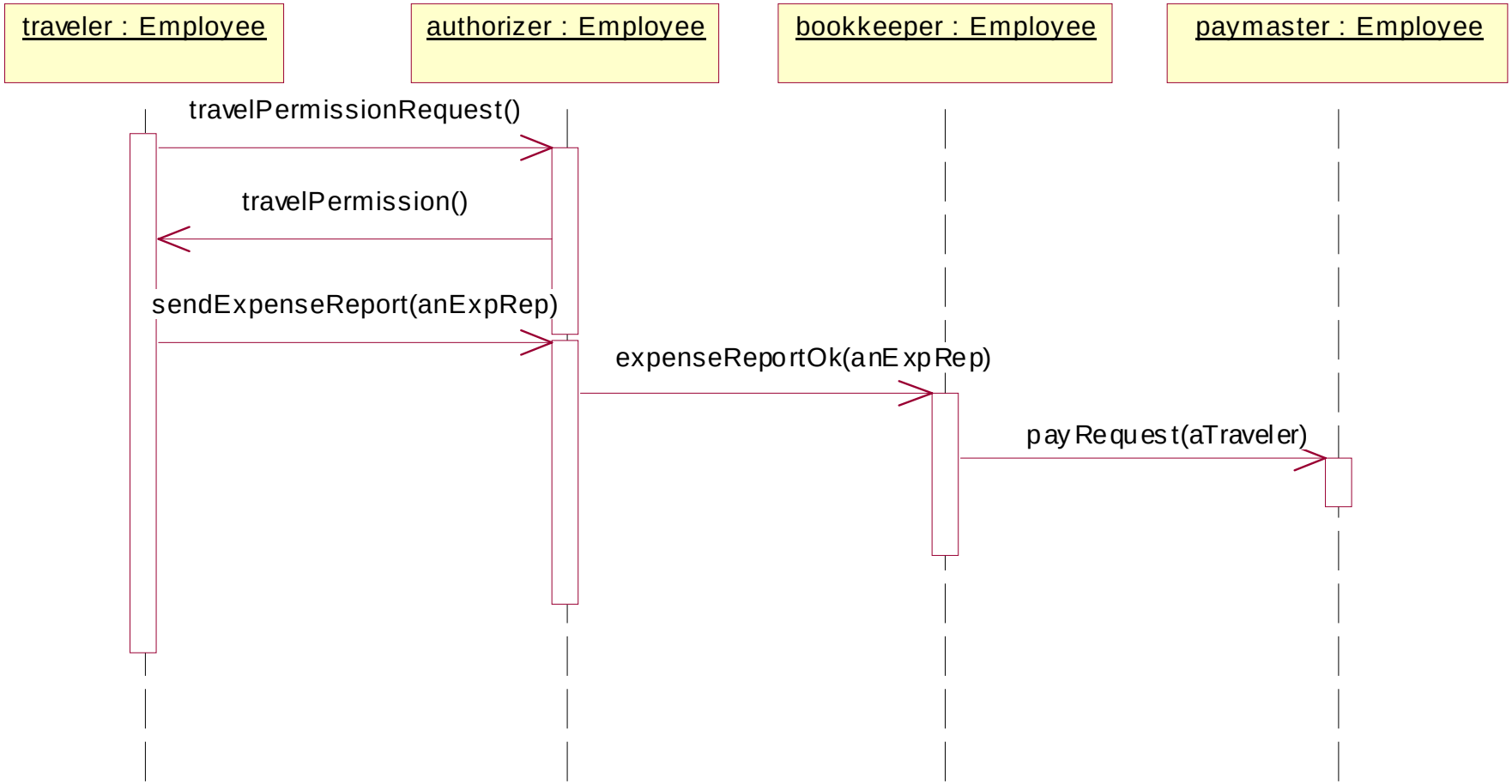


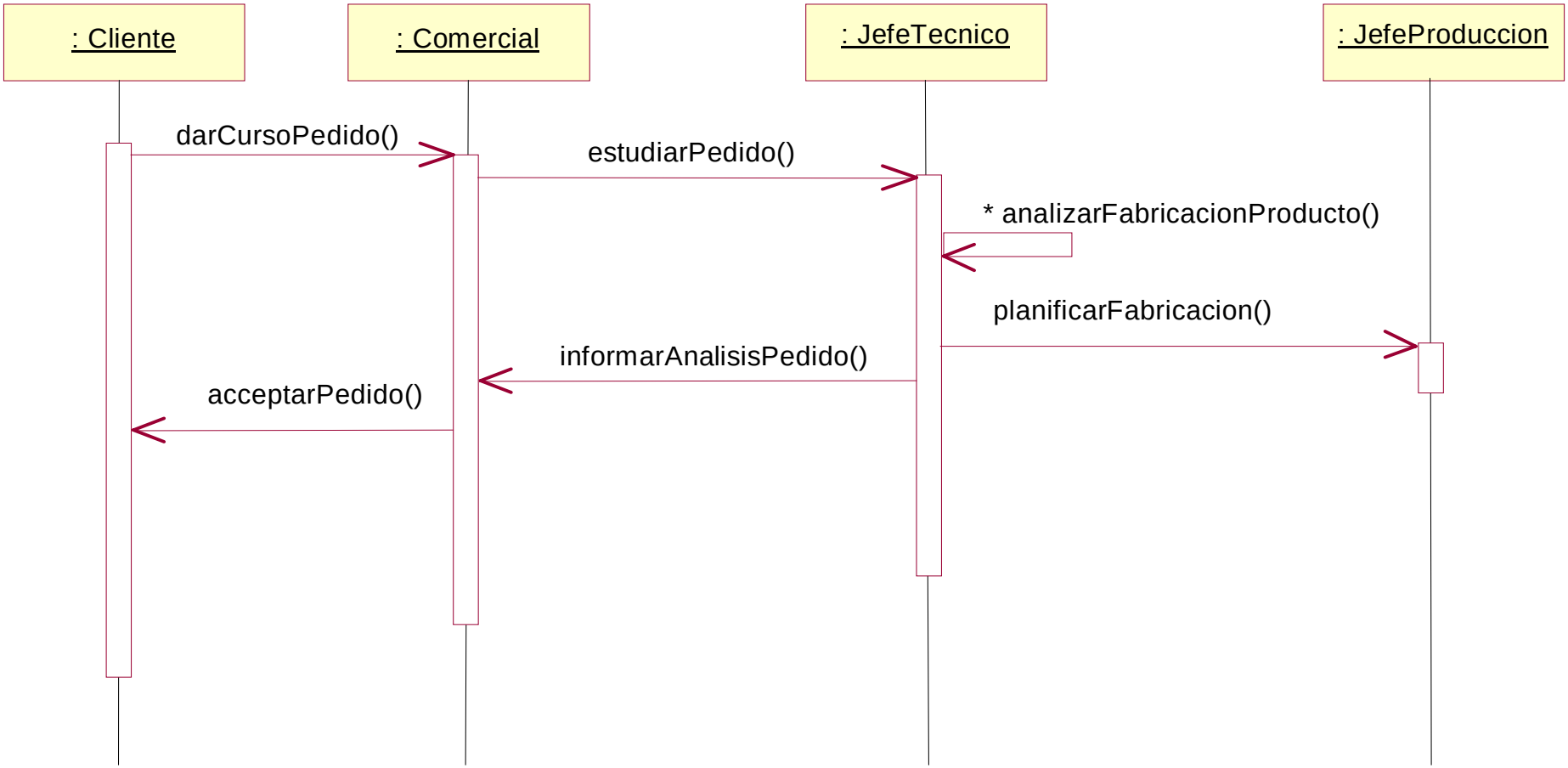
Diagrama de Secuencia



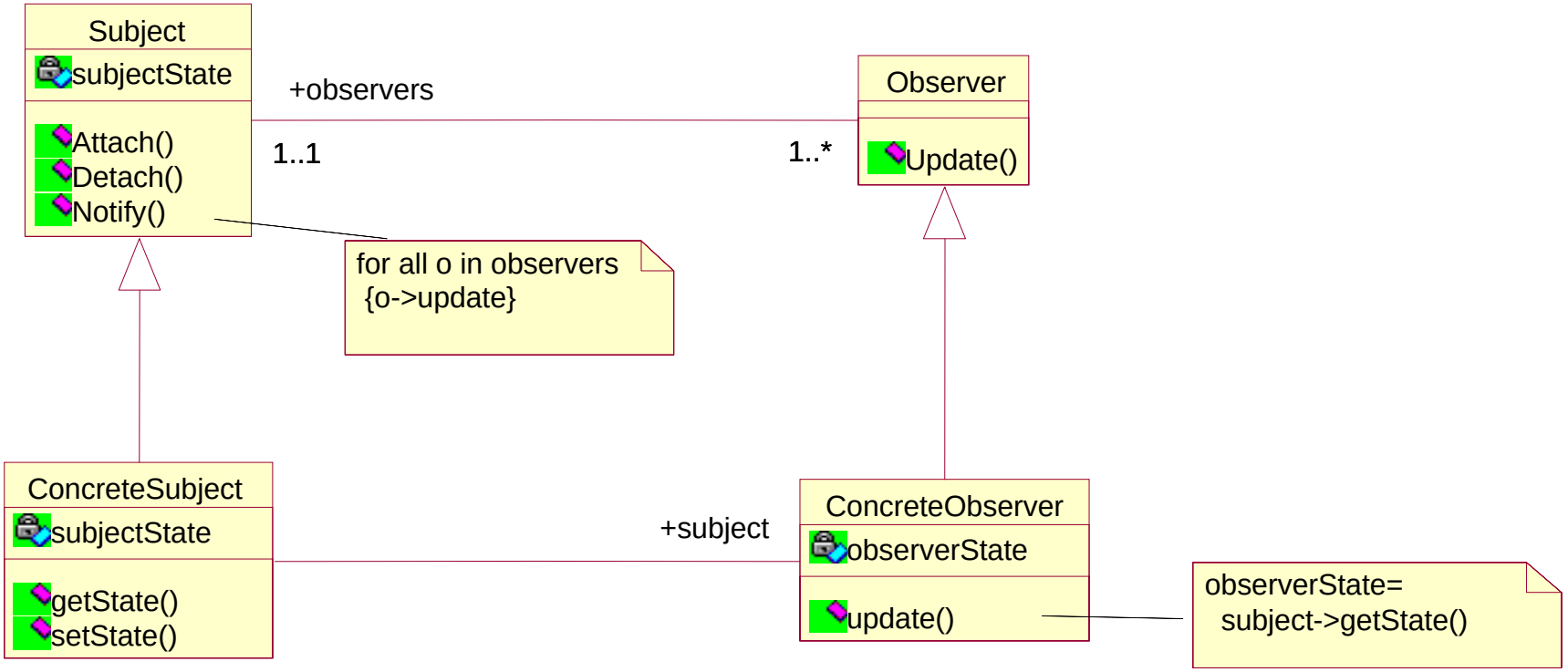
Escenario de un proceso de negocio



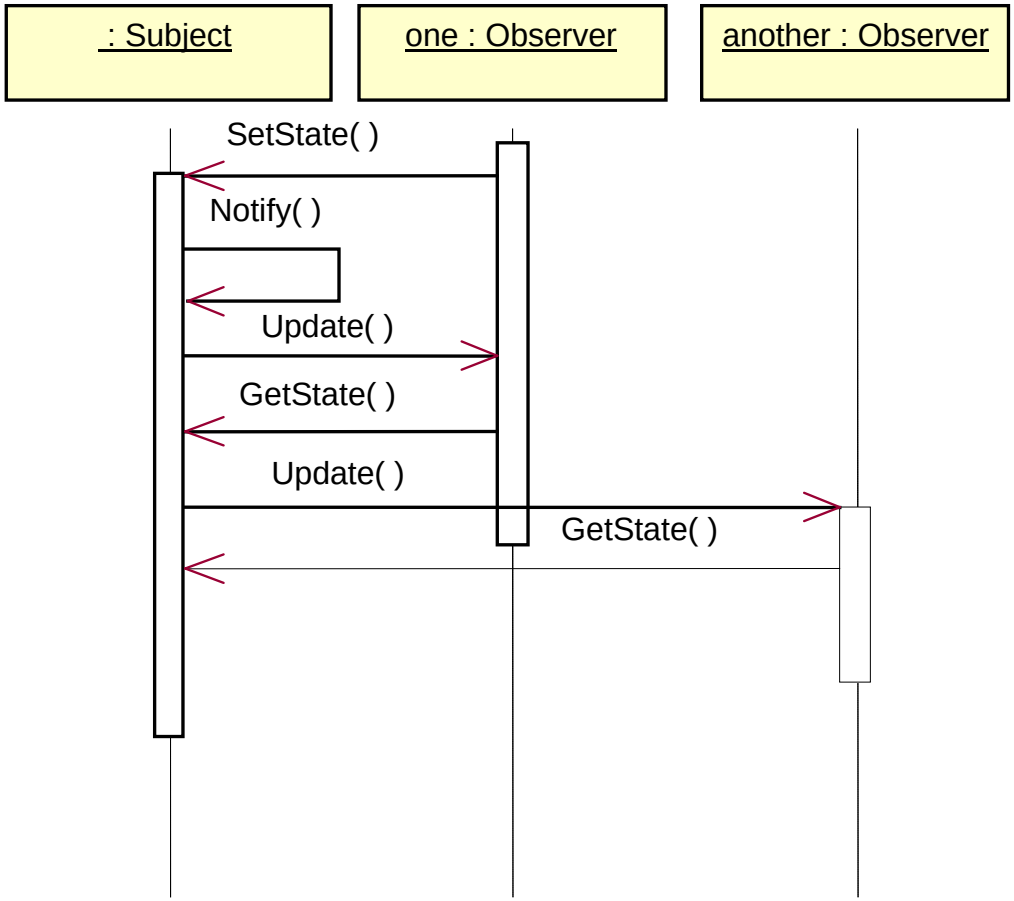
Escenario de un proceso de negocio



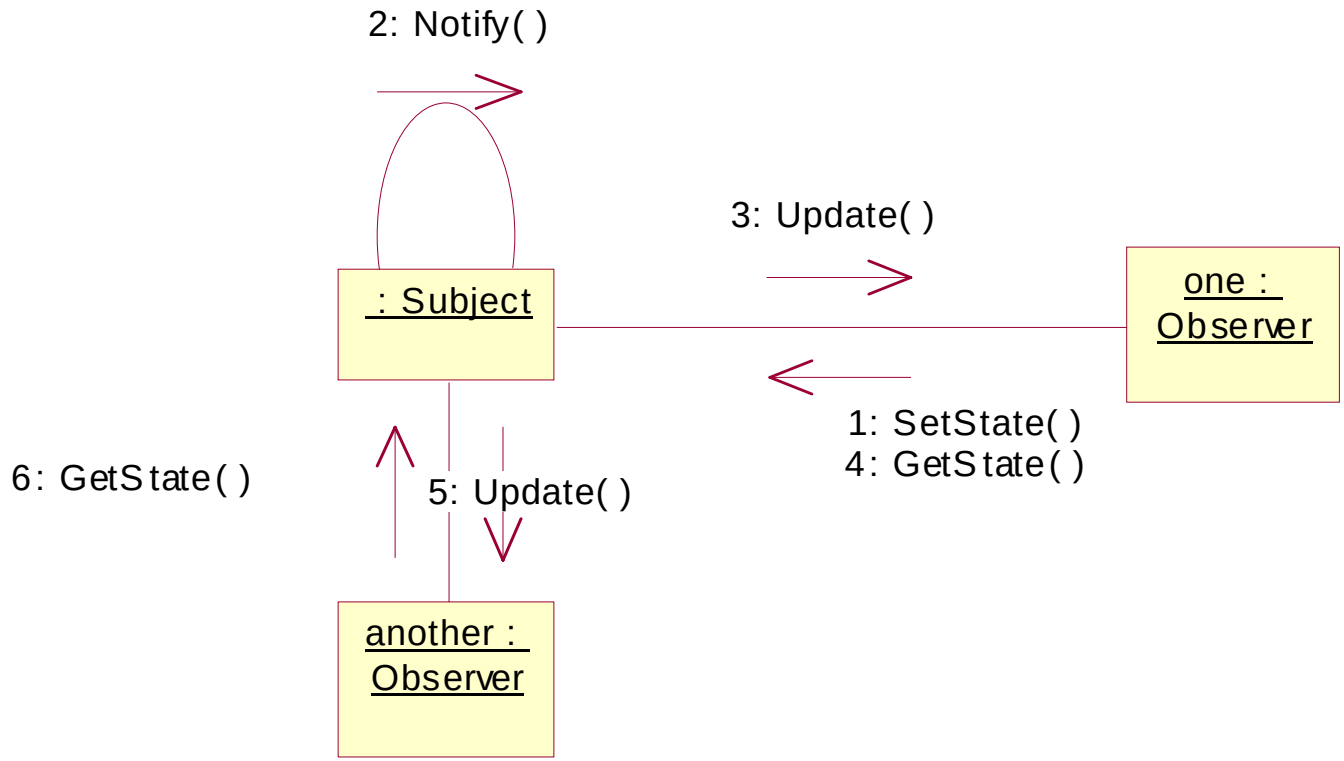
Patrón *Observer*



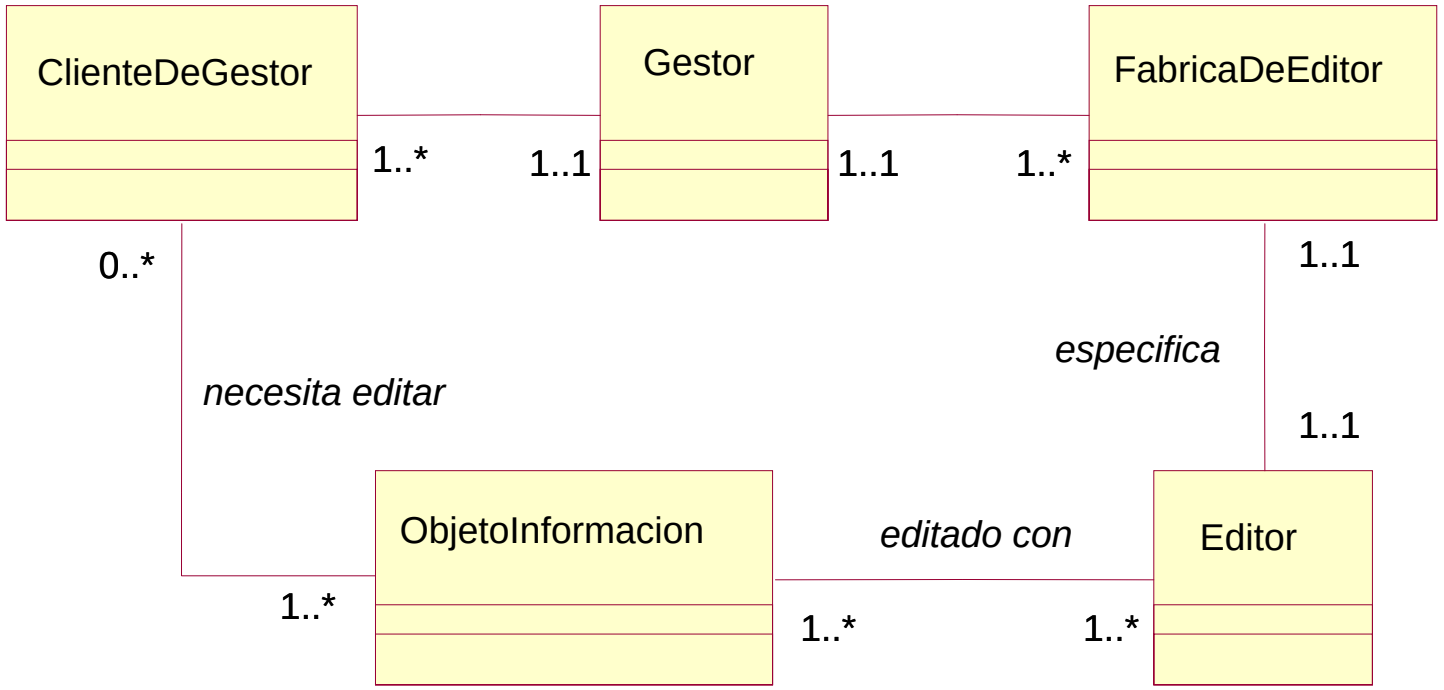
Patrón *Observer*



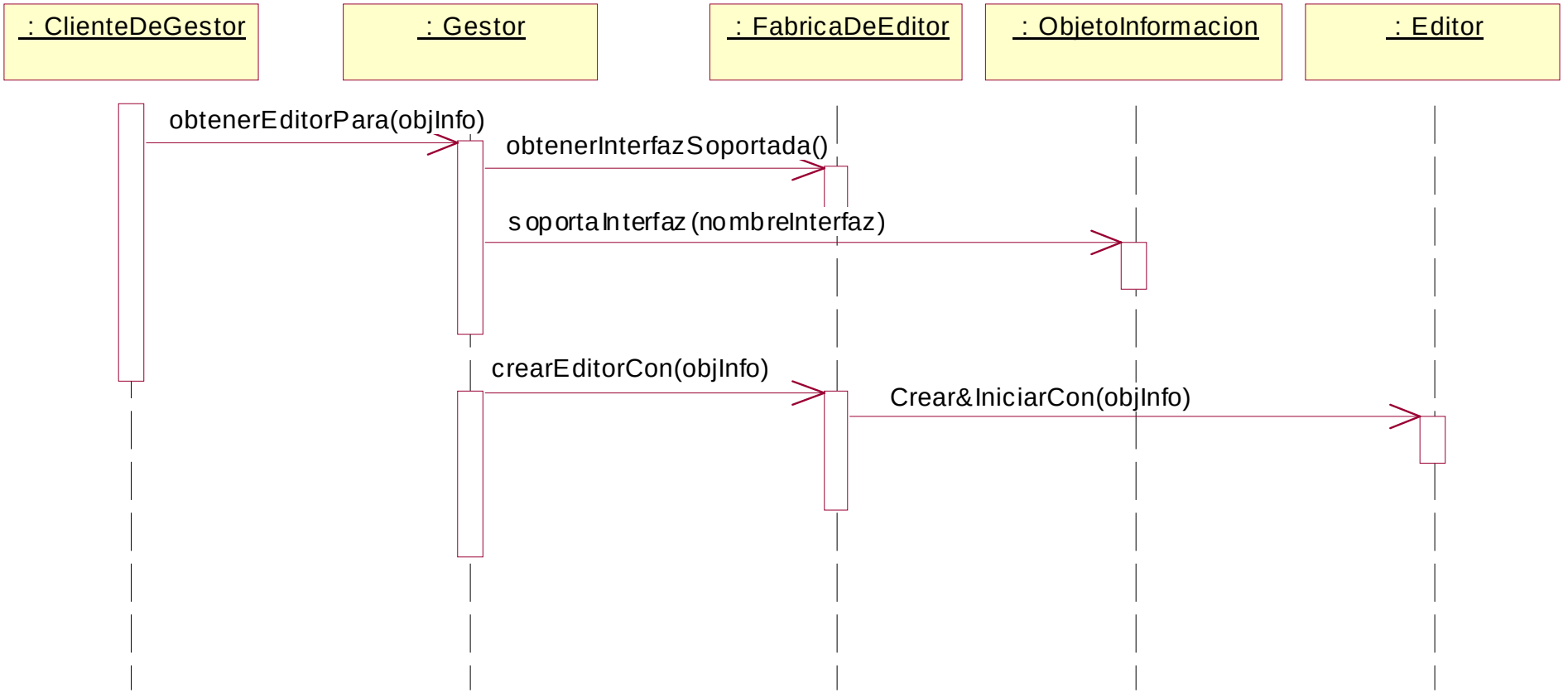
Patrón *Observer*



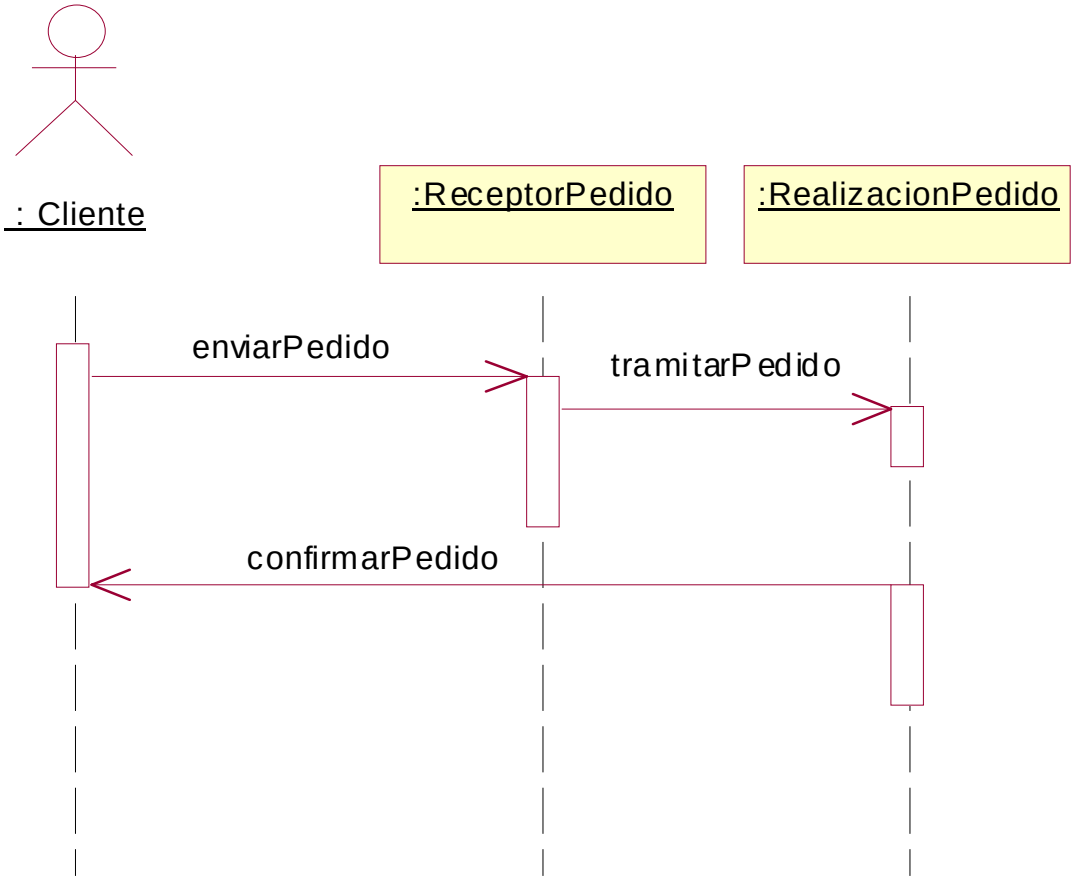
Colaboración



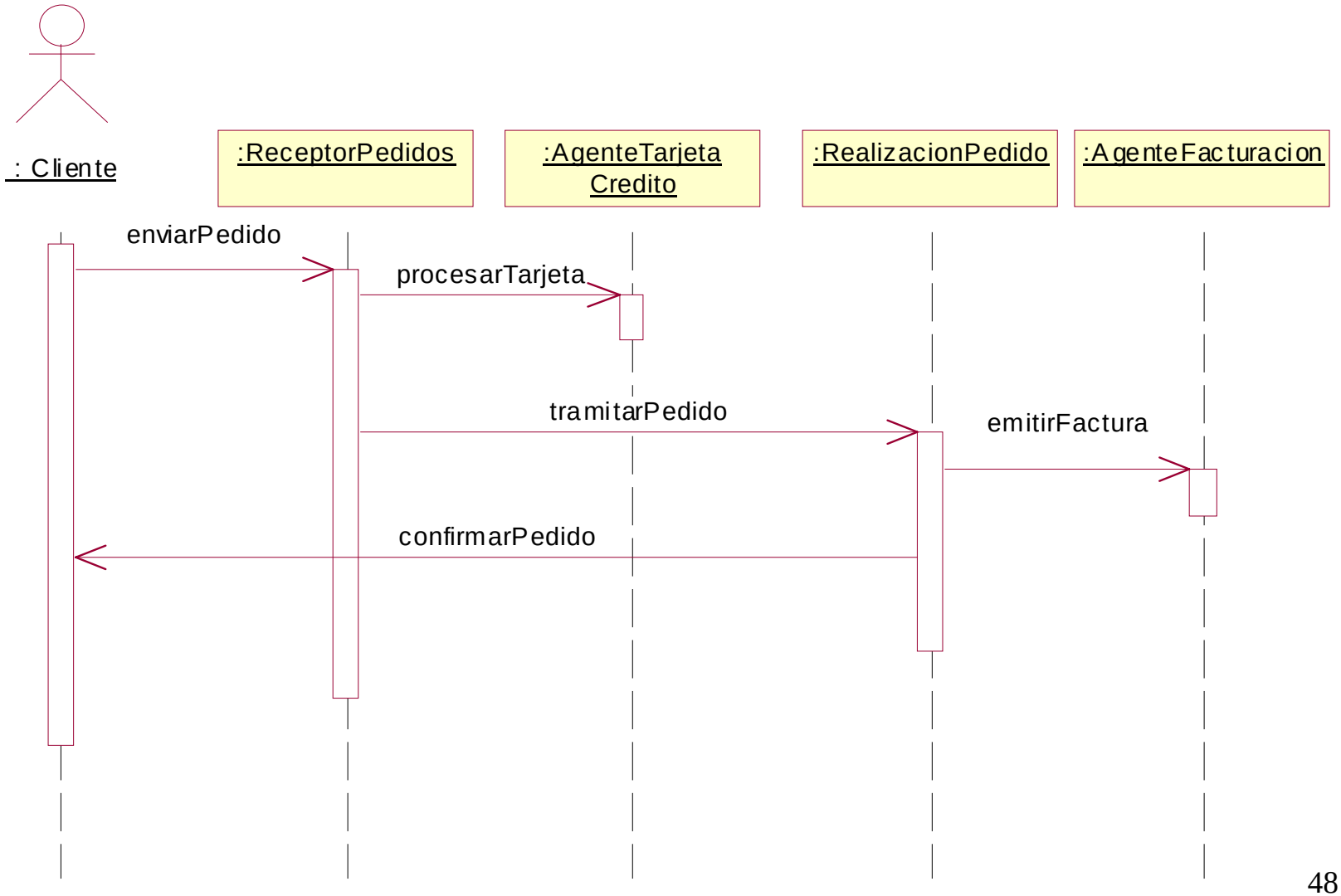
Colaboración



Caso de Uso (Nivel alto de abstracción)



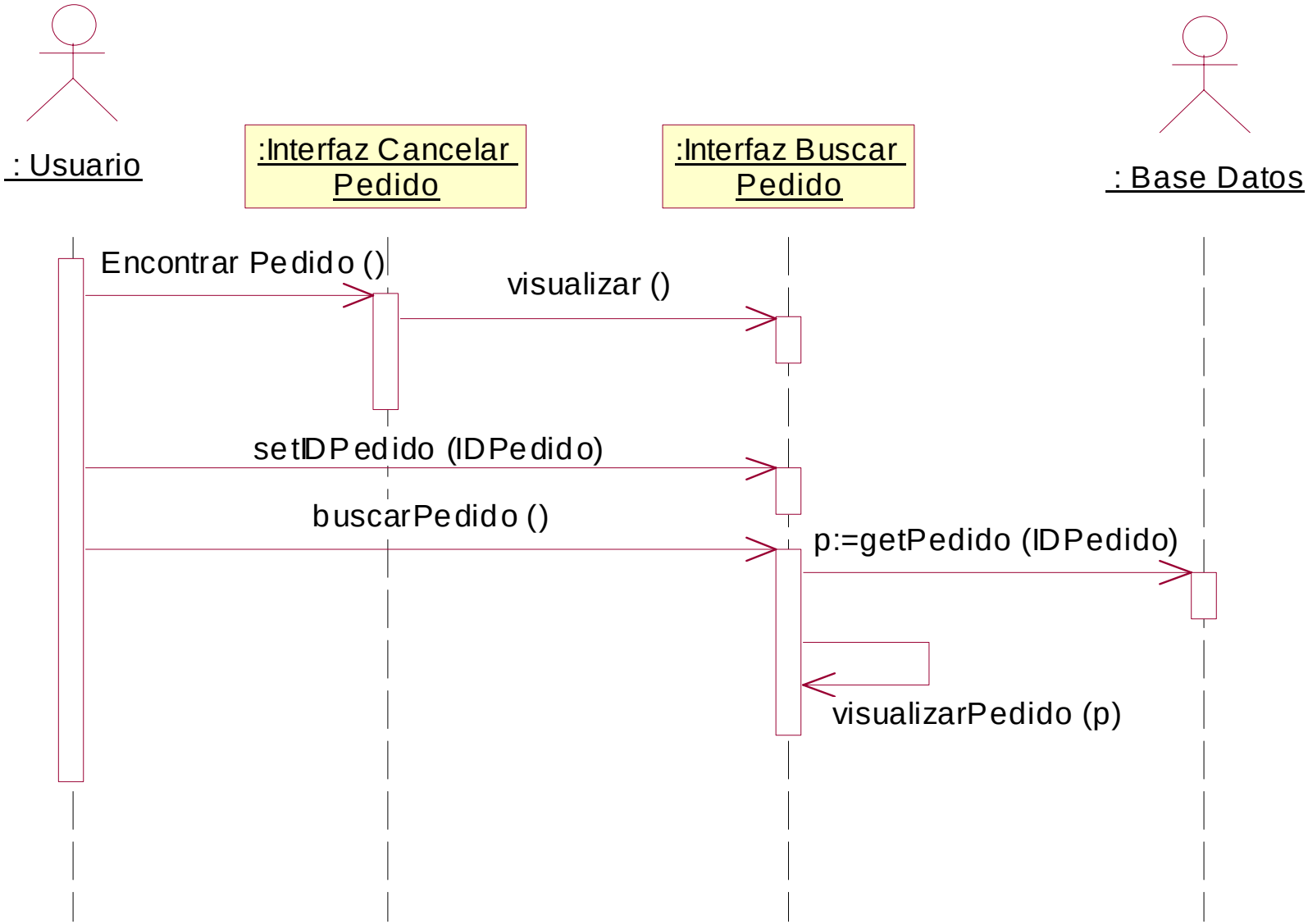
Caso de Uso (Nivel más bajo de abstracción)



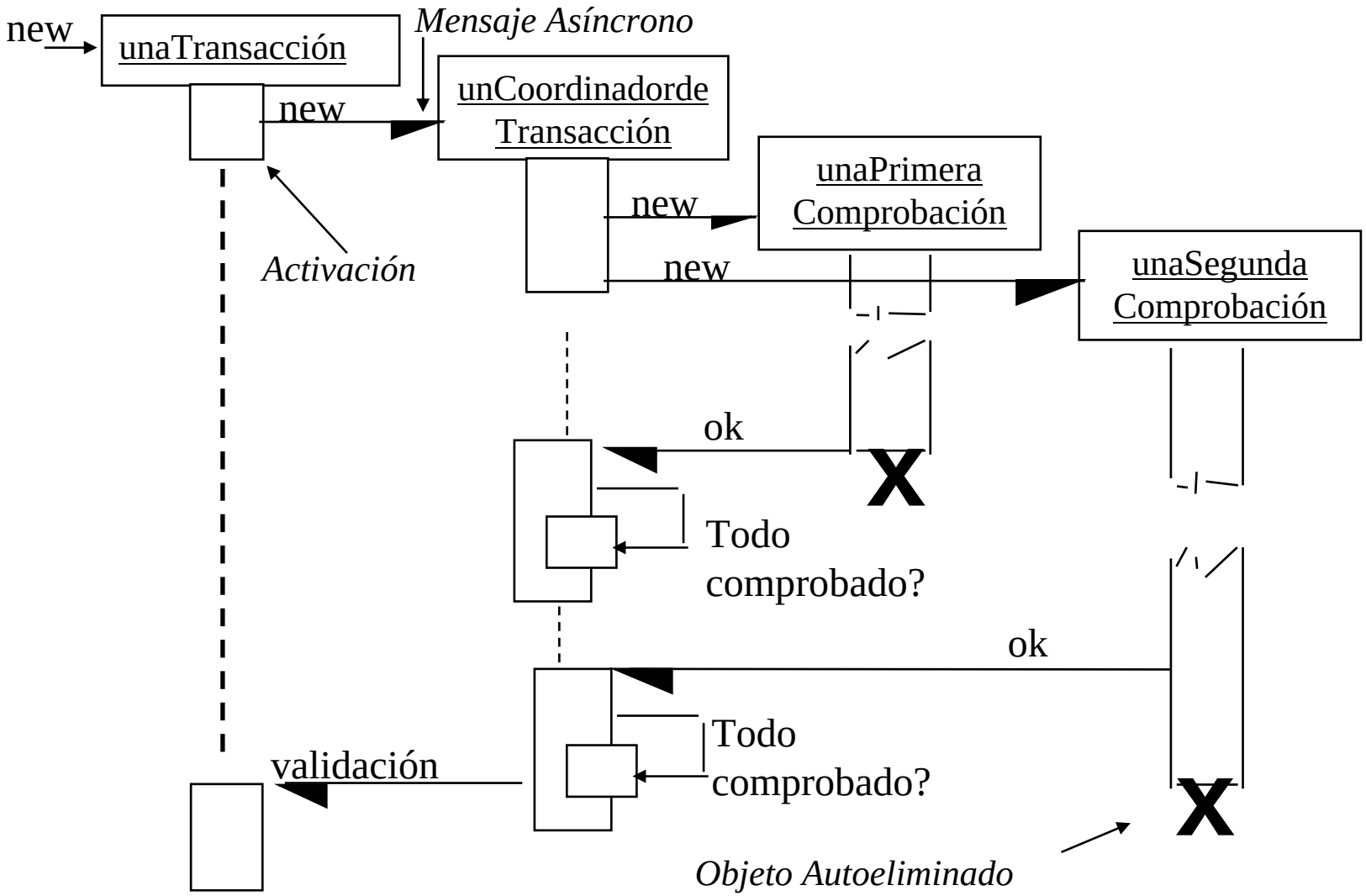
Casos de Uso y Escenarios

Caso de Uso *Buscar Pedido*

1. Se inicia al seleccionar el usuario la opción “*Buscar Pedido”.*
2. El sistema presenta la ventana “Buscar Pedido”.
3. El usuario introduce un ID de un pedido
4. El usuario arranca la búsqueda
5. El sistema realiza una búsqueda en la BD de pedidos.
6. El sistema visualiza datos pedido.


: Usuario
:Interfaz Cancelar Pedido
:Interfaz Buscar Pedido
: Base Datos
Encontrar Pedido ()
visualizar ()
setIDPedido (IDPedido)
buscarPedido ()
p:=getPedido (IDPedido)
visualizarPedido (p)

Diagramas de Secuencia: Procesos concurrentes y activaciones



Diagramas de Secuencia vs. Diagramas de Colaboración

- Equivalencia semántica
- Simples para comportamientos simples.
- Si hay mucho comportamiento condicional, usar diferentes escenarios.
- Diagramas de secuencia muestran mejor el orden en que se ejecutan los mensajes
- Diagramas de colaboración muestran claramente los objetos con que interactúa un determinado objeto.

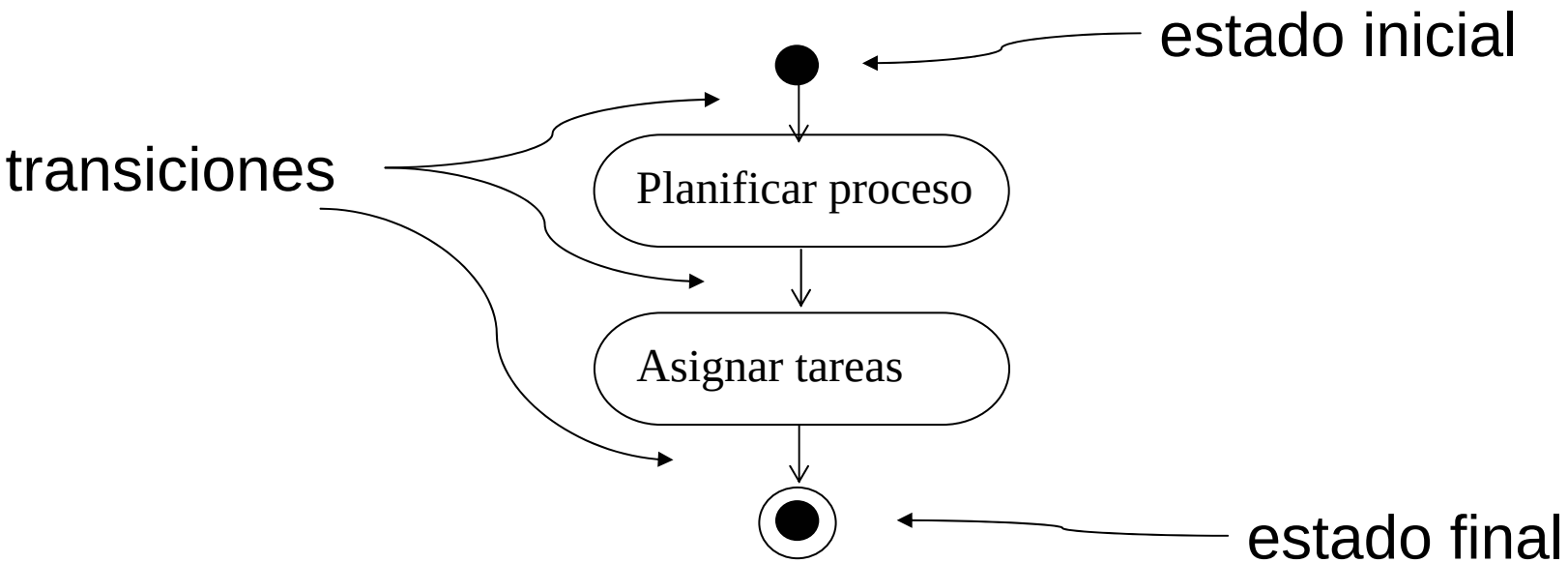
Diagramas de Actividades

- Muestran un flujo de actividades.
- Es un caso especial de máquina de estados.
- Incluye:
  - estados actividad y estados acción
  - transiciones
  - objetos
- Una actividad realiza alguna acción que produce algún cambio en el sistema o retorna un valor.

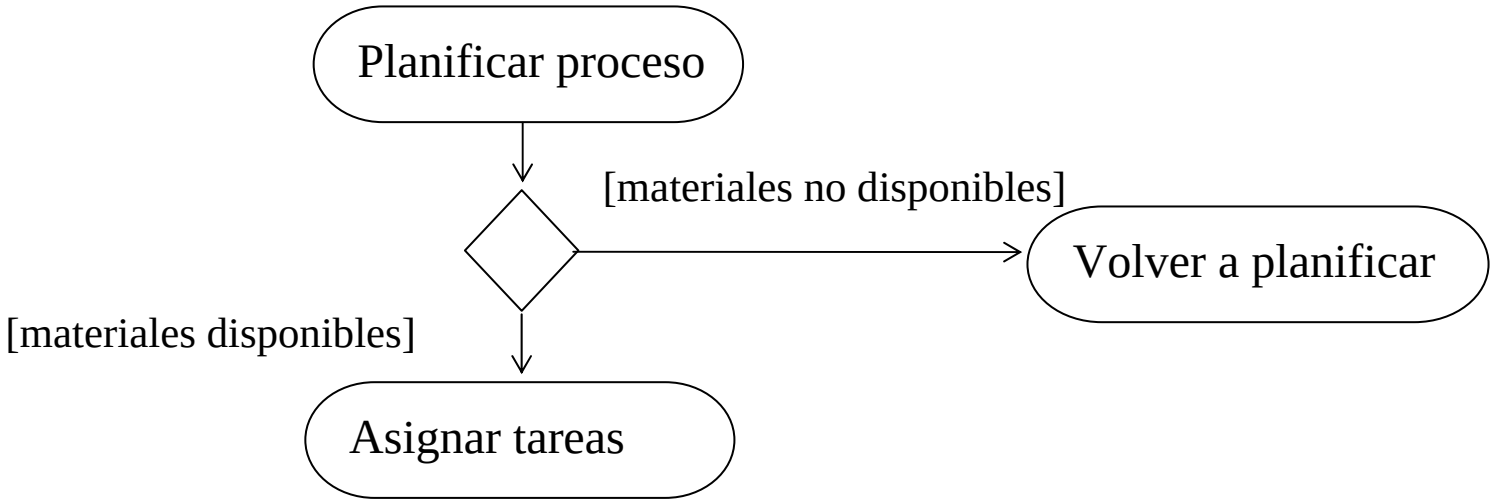
Estados Acción y Estados Actividad

- Un estado acción no se puede descomponer, representa una computación atómica.
- Un estado actividad no es atómico, se compone de otros estados acción y actividad.
- Al entrar se ejecuta la acción o actividad, al terminar el flujo de control pasa a la siguiente acción o actividad.
- Misma notación

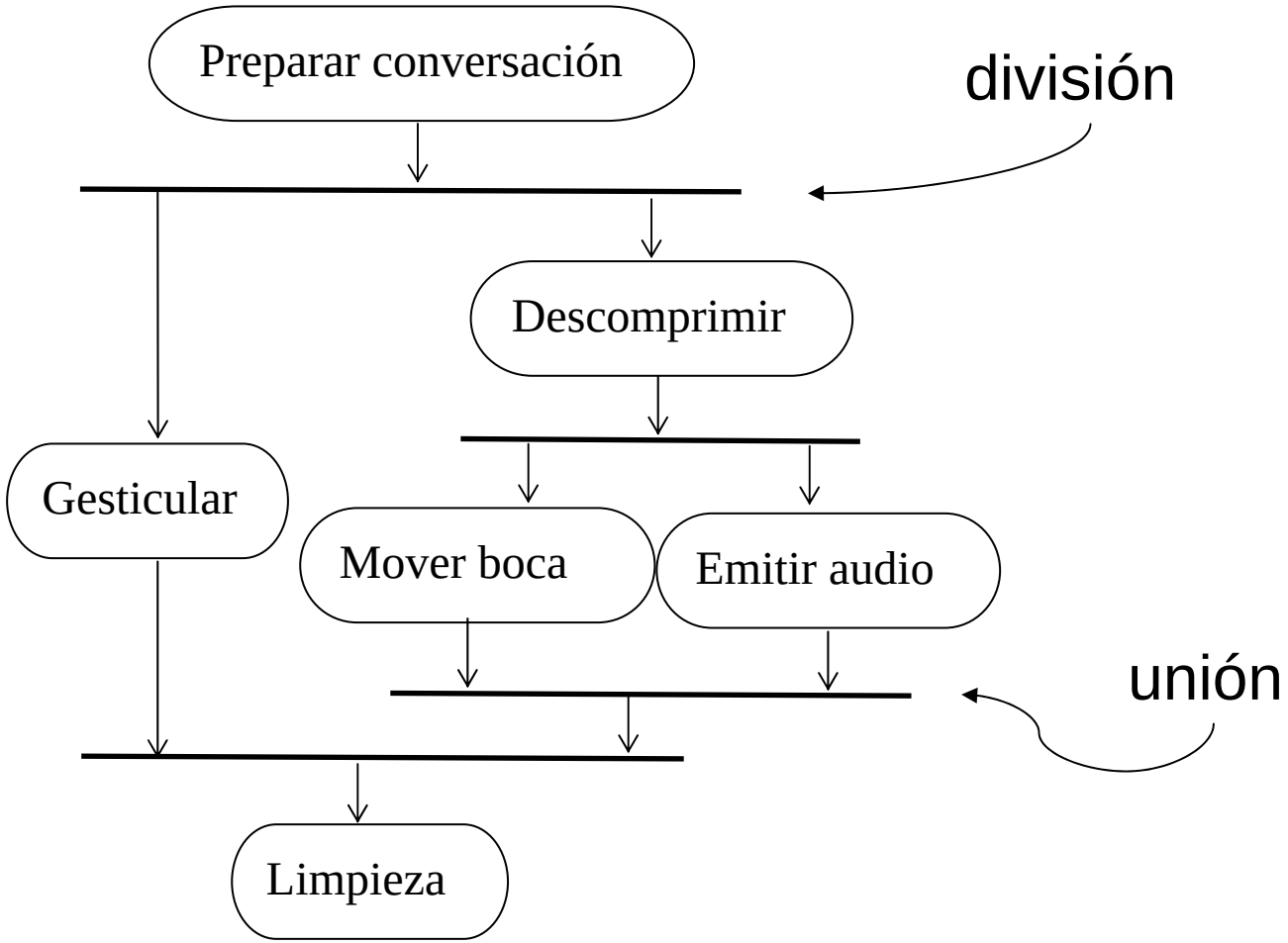
Transiciones

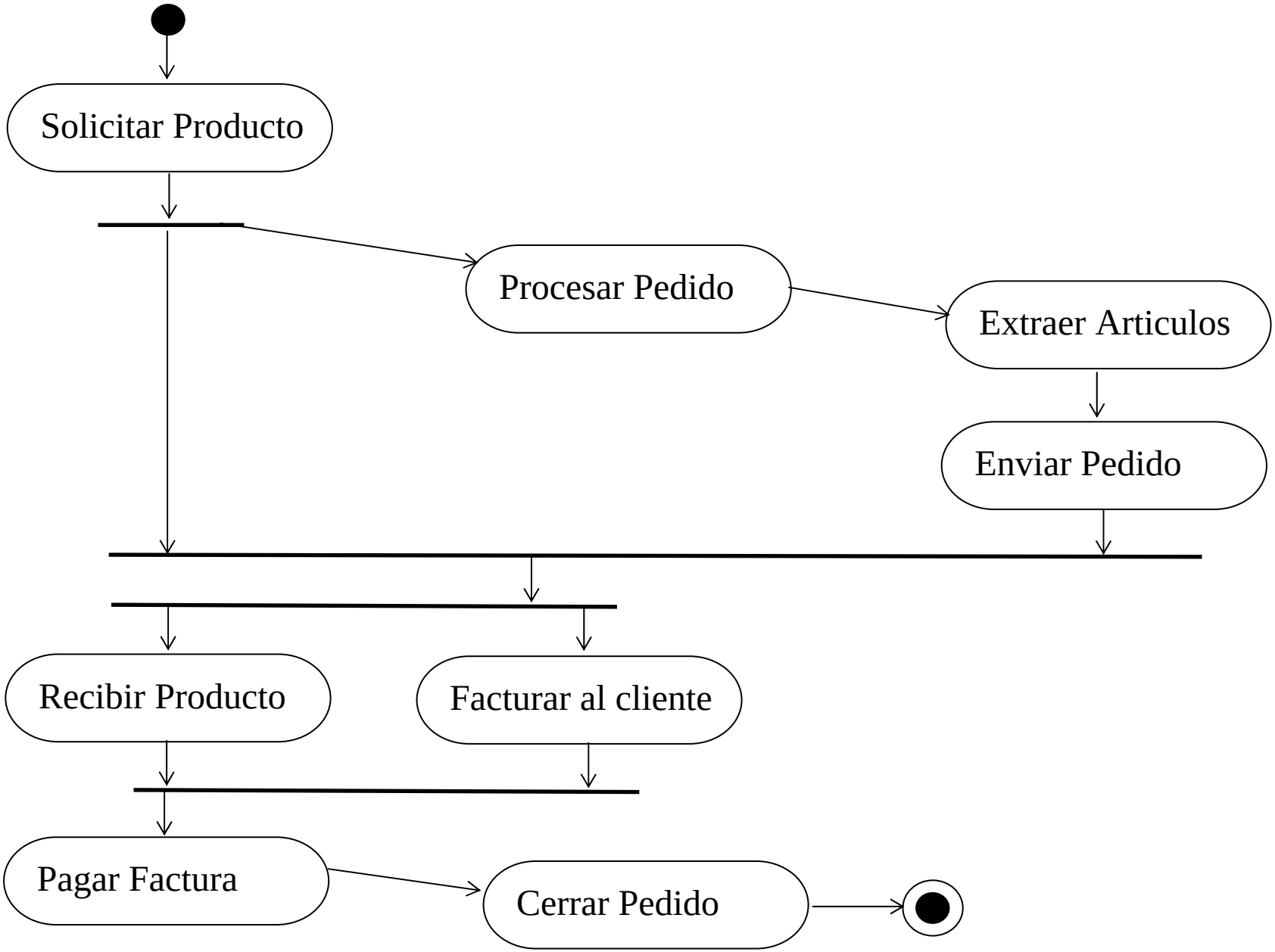


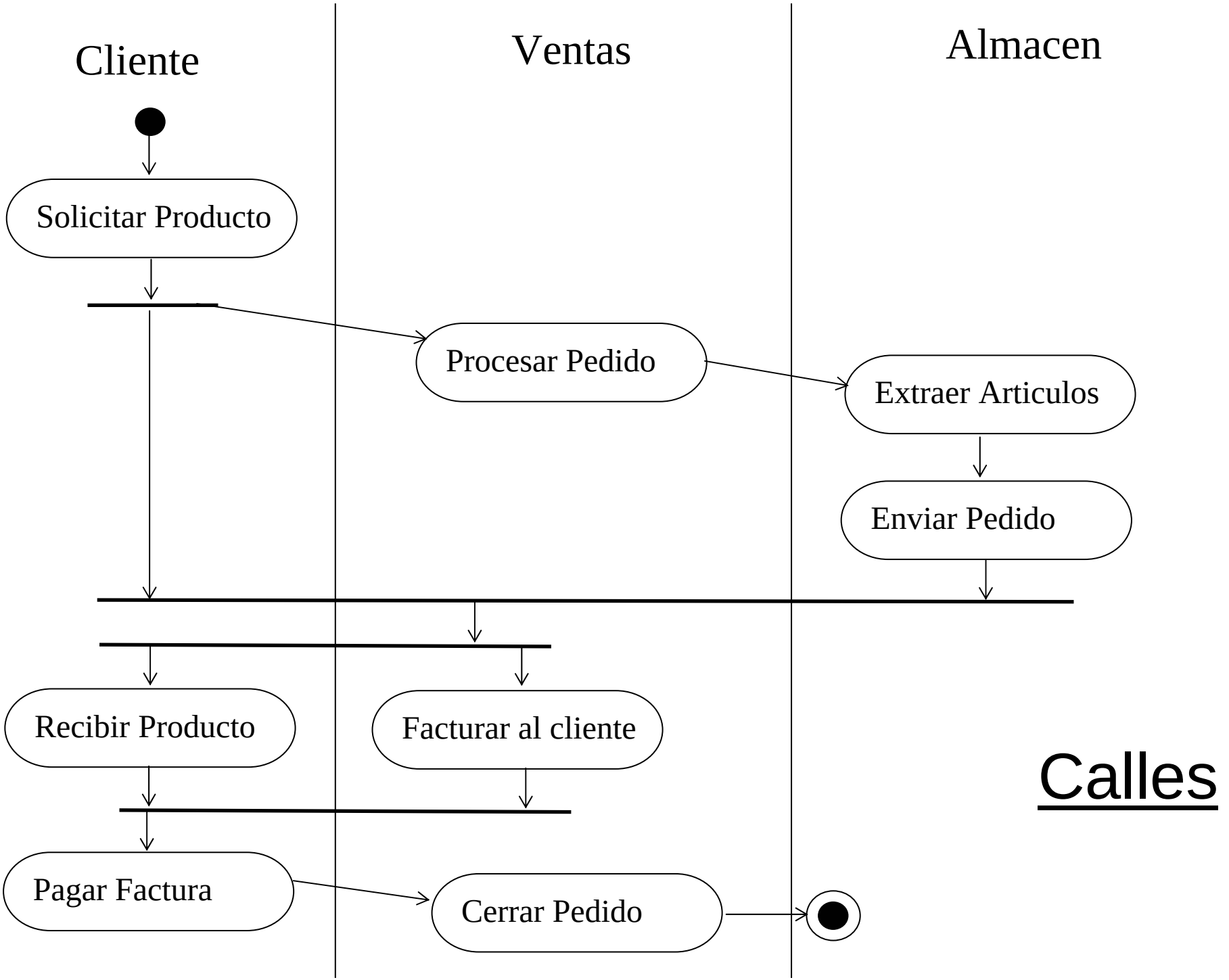
Bifurcación



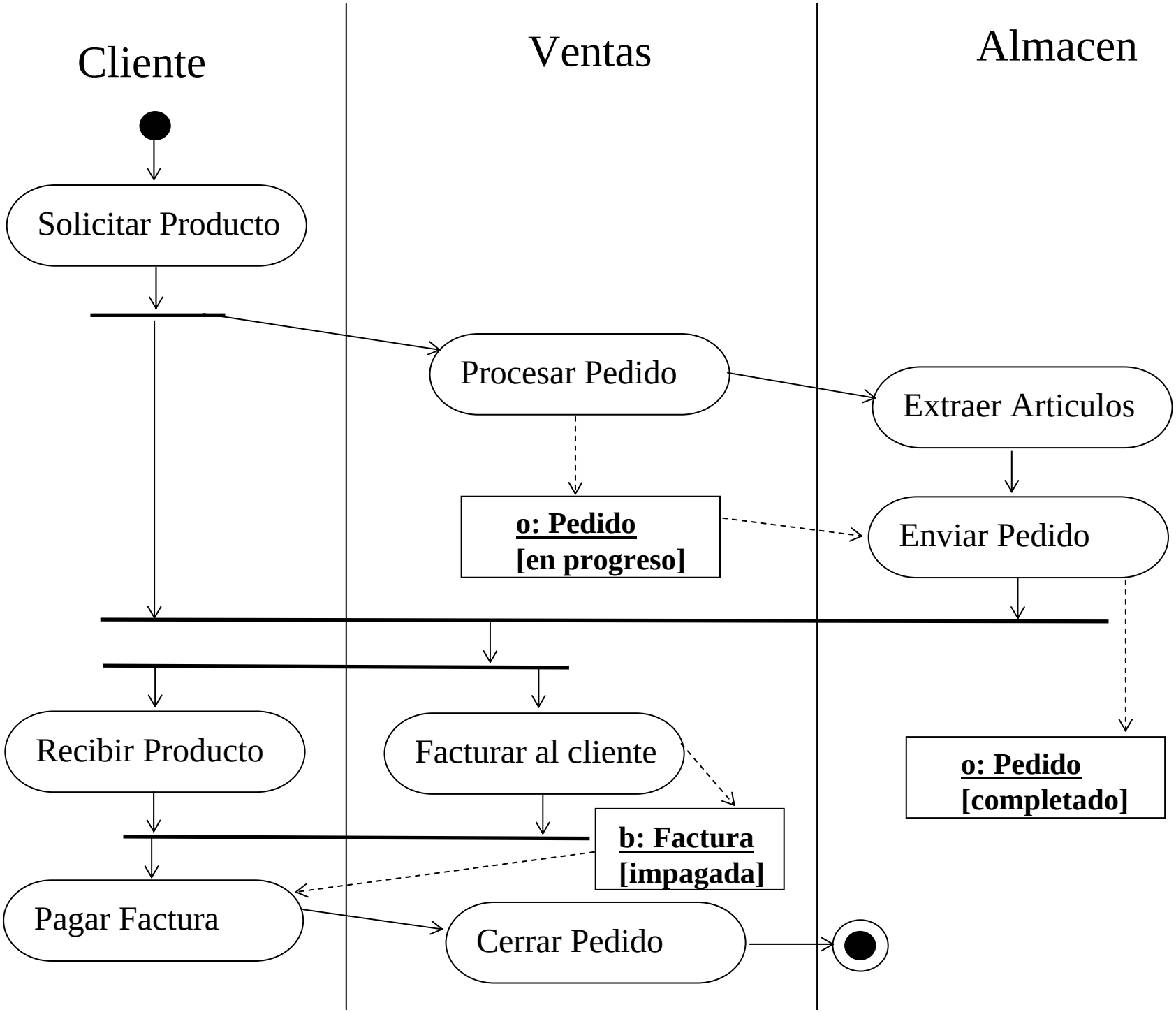
División y Unión



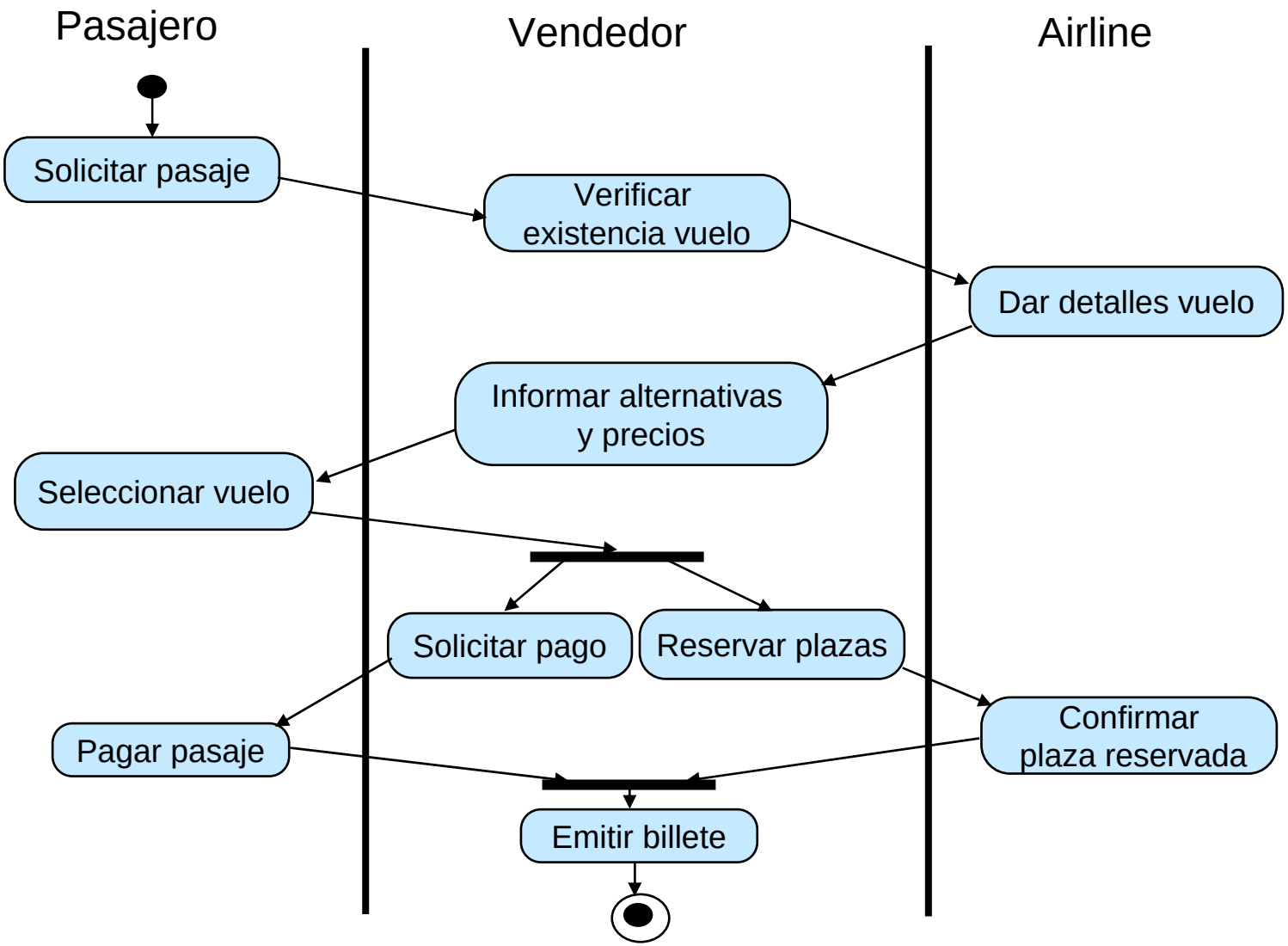

Solicitar Producto
Procesar Pedido
Extraer Articulos
Enviar Pedido
Recibir Producto
Facturar al cliente
Pagar Factura
Cerrar Pedido

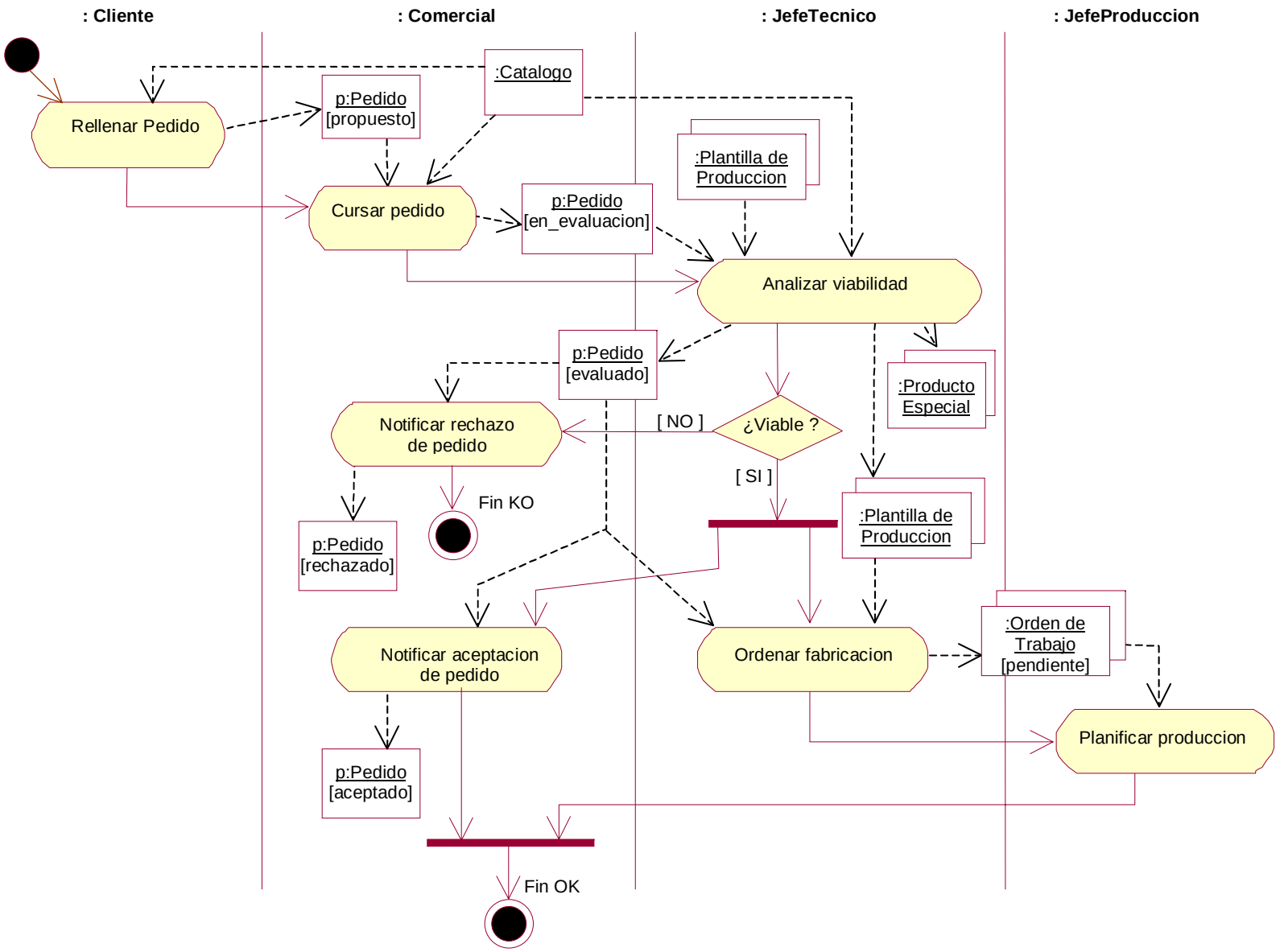

Cliente
Ventas
Almacen
Solicitar Producto
Procesar Pedido
Extraer Articulos
Enviar Pedido
Recibir Producto
Facturar al cliente
Pagar Factura
Cerrar Pedido

Calles


Cliente
Ventas
Almacen
Solicitar Producto
Procesar Pedido
Extraer Articulos
o: Pedido
[en progreso]
Enviar Pedido
Recibir Producto
Facturar al cliente
o: Pedido
[completado]
b: Factura
[impagada]
Pagar Factura
Cerrar Pedido

Ejemplo


Pasajero
Vendedor
Airline
Solicitar pasaje
Verificar existencia vuelo
Dar detalles vuelo
Informar alternativas y precios
Seleccionar vuelo
Solicitar pago
Reservar plazas
Pagar pasaje
Confirmar plaza reservada
Emitir billete


: Cliente
: Comercial
: JefeTecnico
: JefeProduccion
Rellenar Pedido
p:Pedido
[propuesto]
:Catalogo
Cursar pedido
p:Pedido
[en_evaluacion]
:Plantilla de
Produccion
Analizar viabilidad
p:Pedido
[evaluado]
:Producto
Especial
Notificar rechazo
de pedido
[NO]
¿Viable ?
[SI]
Fin KO
p:Pedido
[rechazado]
:Plantilla de
Produccion
Notificar aceptacion
de pedido
Ordenar fabricacion
:Orden de
Trabajo
[pendiente]
Planificar produccion
p:Pedido
[aceptado]
Fin OK

Utilidad de los diagramas de actividades

- Modelado de flujos de trabajo (*workflow*) como son los procesos de negocio (*business processes*).
- Se puede asociar a cualquier elemento de modelado, pero lo más normal es asociarlo a una operación: diagrama de flujo de las acciones.

Diagramas de Estados (*Statecharts*)

- Los s*tatecharts* representan máquinas de estados (autómatas de estados finitos), con estados y transiciones
- Los Statecharts de UML son básicamente los de David Harel, extendidos con características OO
- Puede ser visto como un grafo de estados conectados por transiciones etiquetadas

Statecharts

- Modelan los aspectos dinámicos del sistema
- Cada objeto sigue el comportamiento descrito en el Statechart asociado a su clase
- Cada objeto está en un estado en cierto instante
- El estado está caracterizado parcialmente por los valores de los atributos del objeto
- Los Statecharts de UML son deterministas
- La transición entre estados es instantánea y se debe a la ocurrencia de eventos
- La suposición básica es que una máquina de estados procesa un evento cada vez y termina con todas las consecuencias del evento antes de procesar otro. Si ocurren dos eventos simultáneamente se procesan como si se hubieran producido en cualquier orden, sin pérdida de generalidad

Eventos

- Un evento es un acontecimiento que ocupa un lugar en el tiempo y espacio.
- Un evento es un estímulo que dispara una transición en una máquina de estados.
- Eventos externos vs. Eventos internos.
- Tipos de eventos:
  - **Señales (excepciones)**
  - **Llamadas**
  - **Paso de tiempo**
  - **Cambio de estado**

Tipos de eventos

llamada: la recepción de una petición para invocar una operación. Normalmente un evento de llamada es modelado como una operación del objeto receptor, manejado por un método del receptor y se implementa como una acción o transición de la máquina de estados

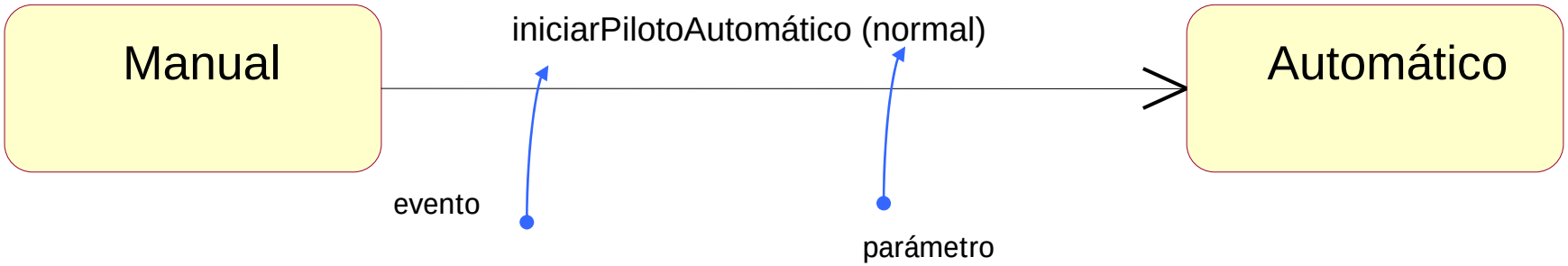
señal: la recepción de una señal, que es una entidad a la que se ha dado nombre explícitamente (clase estereotipada), prevista para la comunicación explícita - y asíncrona- entre objetos. Es enviada por un objeto a otro objeto o conjunto de objetos. Las señales con nombre que puede recibir un objeto se modelan designándolas en un compartimento extra de la clase de ese objeto. Normalmente una señal es manejada por la máquina de estados del objeto receptor y puede disparar una transición en la máquina de estados

tiempo: representa el paso del tiempo (ocurrencia de un tiempo absoluto respecto de un reloj real o virtual o el paso de una cantidad de tiempo dada desde que un objeto entra en un estado). Palabra clave **after:** after (2 segundos); *after 1 ms desde la salida de devInactivo*

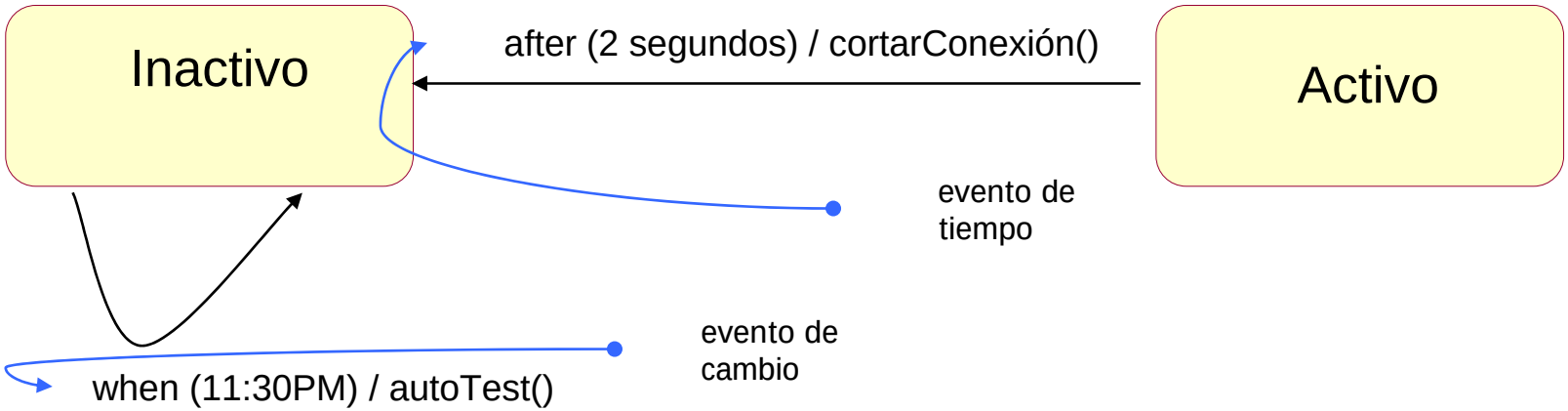
cambio: evento que representa un cambio en el estado o el cumplimiento de alguna condición. Palabra clave **when**, seguida de una expresión booleana, que puede ser de tiempo o de otra clase: when (hora = 11:30); when (altitud < 1000)

Tipos de eventos

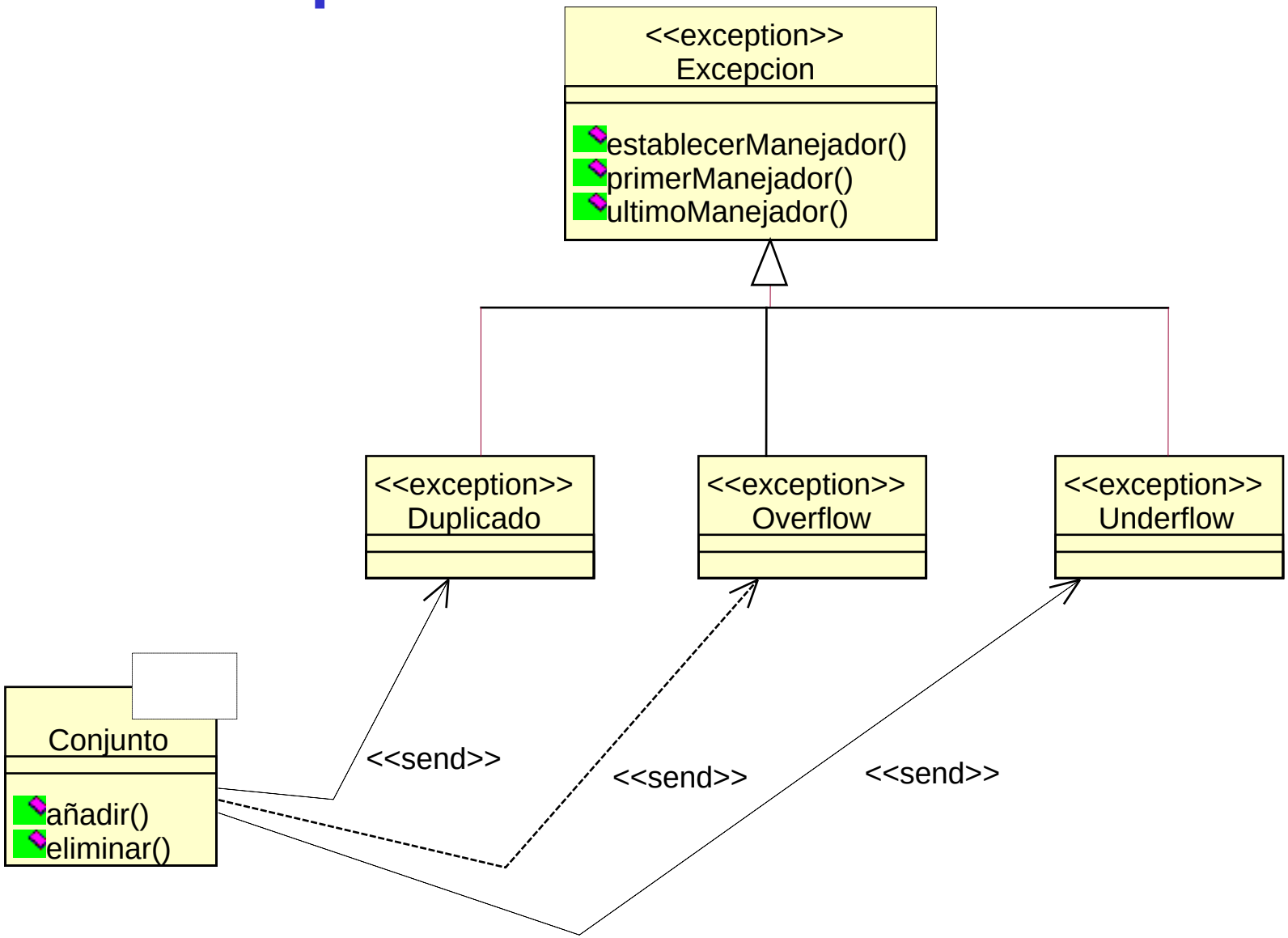
- Evento de llamada (se representa igual que el de señal)



- Eventos de tiempo y de cambio



Modelado Excepciones



Estados

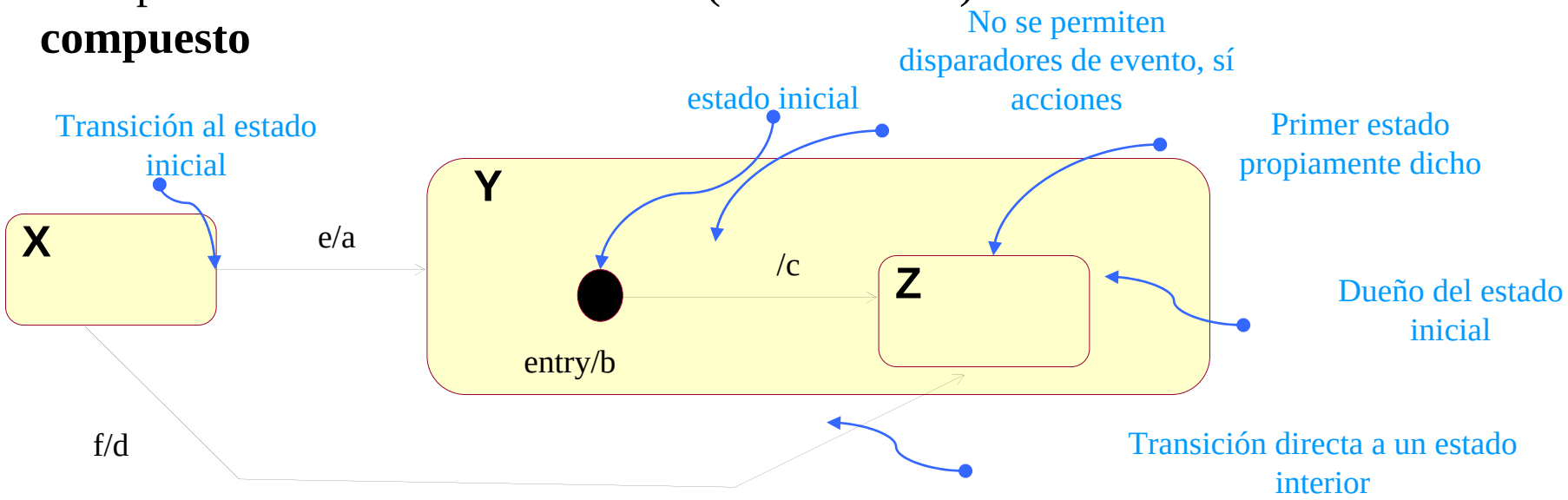
- Un estado es una situación en la vida de un objeto en la que satisface cierta condición, realiza alguna actividad o espera algún evento.
- Elementos de un estado
  - Nombre
  - Acciones entrada/salida
  - Transiciones internas
  - Subestados
  - Eventos diferidos

Estados


Rastreando
entry/ activarModo(enRastreo)
exit / activarModo(noRastreo)
nuevoObjetivo/rastreador.adquirir
do / seguirObjetivo
autotest / defer
acción entrada
transición interna
evento diferido
acción salida
actividad

Estado inicial

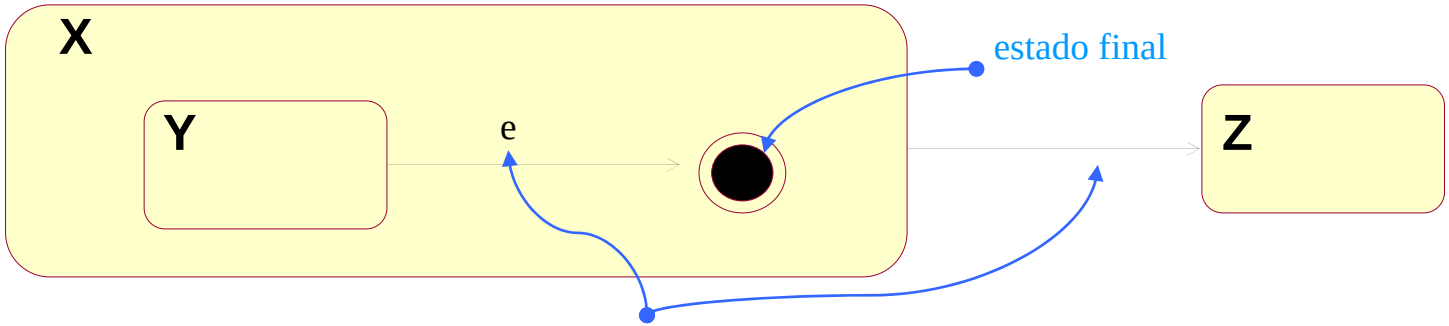
- Se trata de un **pseudoestado** que indica el punto de partida por defecto para una transición cuyo destino es el límite de un estado compuesto
- El estado inicial del estado de nivel más alto representa la **creación** de una nueva **instancia** de la clase
- **Sólo** puede haber **un estado inicial** (directamente) dentro de **cada estado compuesto**



Si estando en X se produce e, la transición pasa al estado Y. Se ejecuta la acción de entrada b y el estado inicial queda activado. La transición saliente se dispara inmediatamente, ejecutando la acción c y pasando al estado Z. Si estando en X se produce f, se ejecuta d, se ejecuta b, al entrar a Y, y se pasa a Z, sin ejecutar c, pues el estado inicial ya no está implicado

Estado final

- Estado **especial** dentro de un **estado compuesto** que, cuando está **activo**, **indica** que la **ejecución** del estado compuesto ha **terminado y que una transición de finalización** que sale del estado compuesto está **activada**.
- Un estado final **no es un pseudoestado**. Puede ser activado por un periodo de tiempo, a diferencia del inicial que transita inmediatamente a su sucesor, p.ej., mientras espera la terminación de otros subestados concurrentes en el mismo estado compuesto
- Las transiciones entrantes son transiciones normales



El evento e causa la transición al estado final, que causa la transición de finalización hacia Z

Estado final

- **Sólo** puede ocurrir **un estado final** (directamente) dentro de un estado compuesto. El símbolo de estado final puede repetirse dentro de un estado, pero cada copia representa el mismo estado final.
- Si un objeto alcanza su estado final de nivel superior, la máquina de estados termina y **se destruye el objeto**
- Es posible que **no haya estado final**, indicando una actividad continua (común por ejemplo en sistemas empotrados)

Transiciones

- Una transición de un estado A a un estado B, se produce cuando se origina el evento asociado y se satisface la condición especificada, en cuyo caso se ejecuta la acción de salida de A, la acción de entrada a B y la acción asociada a la transición.
- Elementos de una transición:
  - **Estados origen y destino**
  - **Evento de disparo**
  - **Condición de guarda**
  - **Acción**

Elementos de una transición

estado origen

- la transición se disparará si, estando en el estado origen, se produce el evento de disparo y si la condición de guarda, si la hay, se satisface

estado destino

- estado activo tras completarse la transición

evento de disparo.

- cuando se produce un evento, afecta a todas las transiciones que lo contienen en su etiqueta. Todas las apariciones de un evento en la misma máquina de estados deben tener la misma signatura

condición de guarda

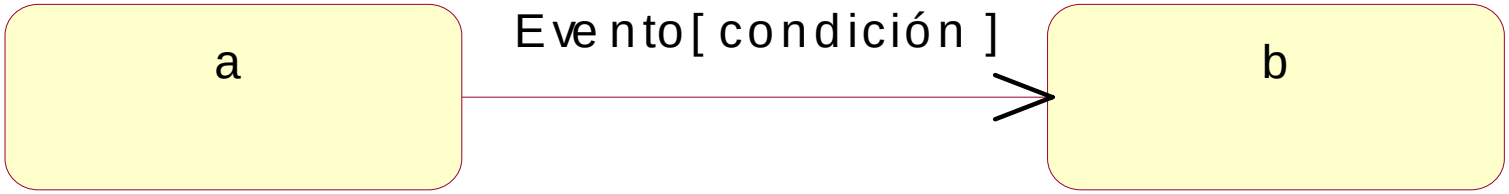
- expresión booleana. Si es falsa, la transición no se dispara, y si no hay otra transición etiquetada con el mismo evento que pueda dispararse, éste se pierde

acción

- computación atómica ejecutable. Puede incluir llamadas a operaciones sobre el objeto que contiene la máquina de estados (o sobre otros visibles), creación o destrucción de objetos, o envío de una señal a otro objeto

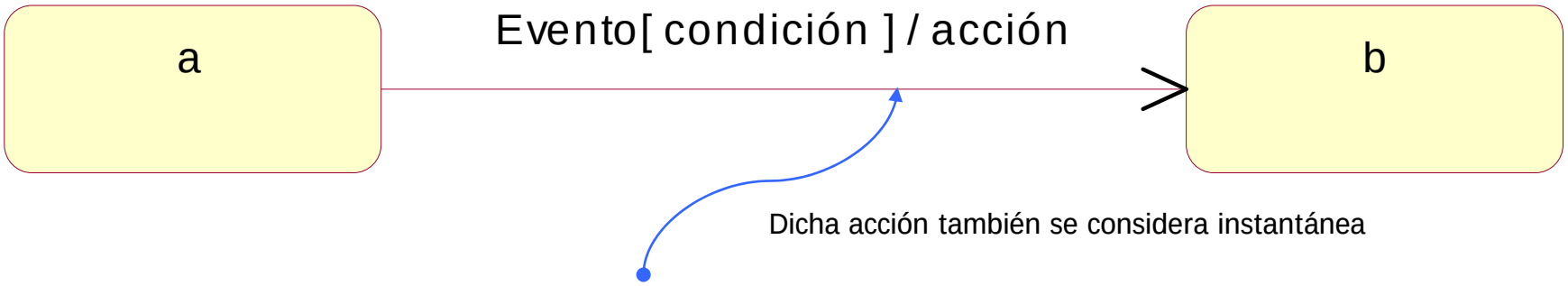
Guardas en una transición

- Las condiciones o guardas permiten condicionar la transición:



Acciones en una transición

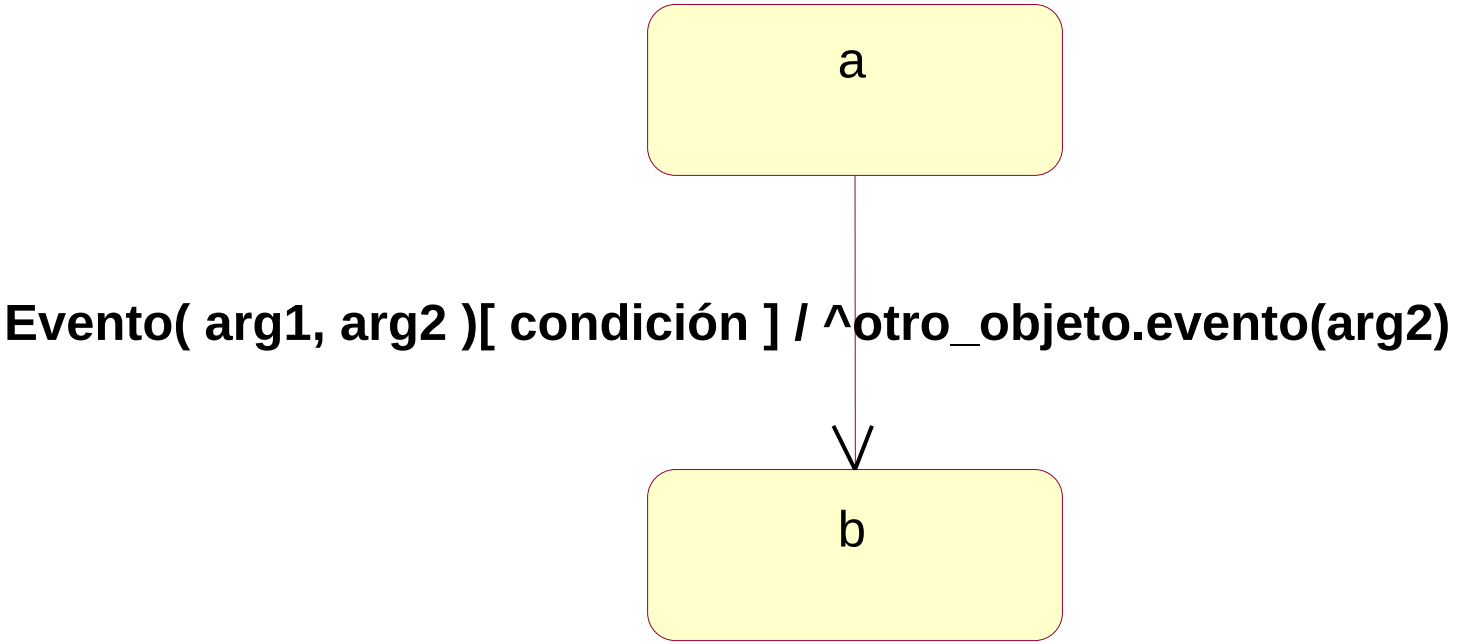
- Podemos especificar la ejecución de una acción como consecuencia de la transición:



Acción: computación atómica ejecutable que produce un cambio en el estado del modelo o que devuelve un valor. Puede ser realizada mediante el envío de un mensaje a un objeto provocando la modificación de un enlace o del valor de un atributo en un objeto. Puede incluir llamadas a operaciones (sobre el objeto que contiene la máquina de estados, así como sobre otros objetos visibles), creación o destrucción de objetos, o el envío de una señal a un objeto

Acciones en una transición

- Podemos especificar el envío de un evento a otro objeto como consecuencia de la transición:



Máquina de estados

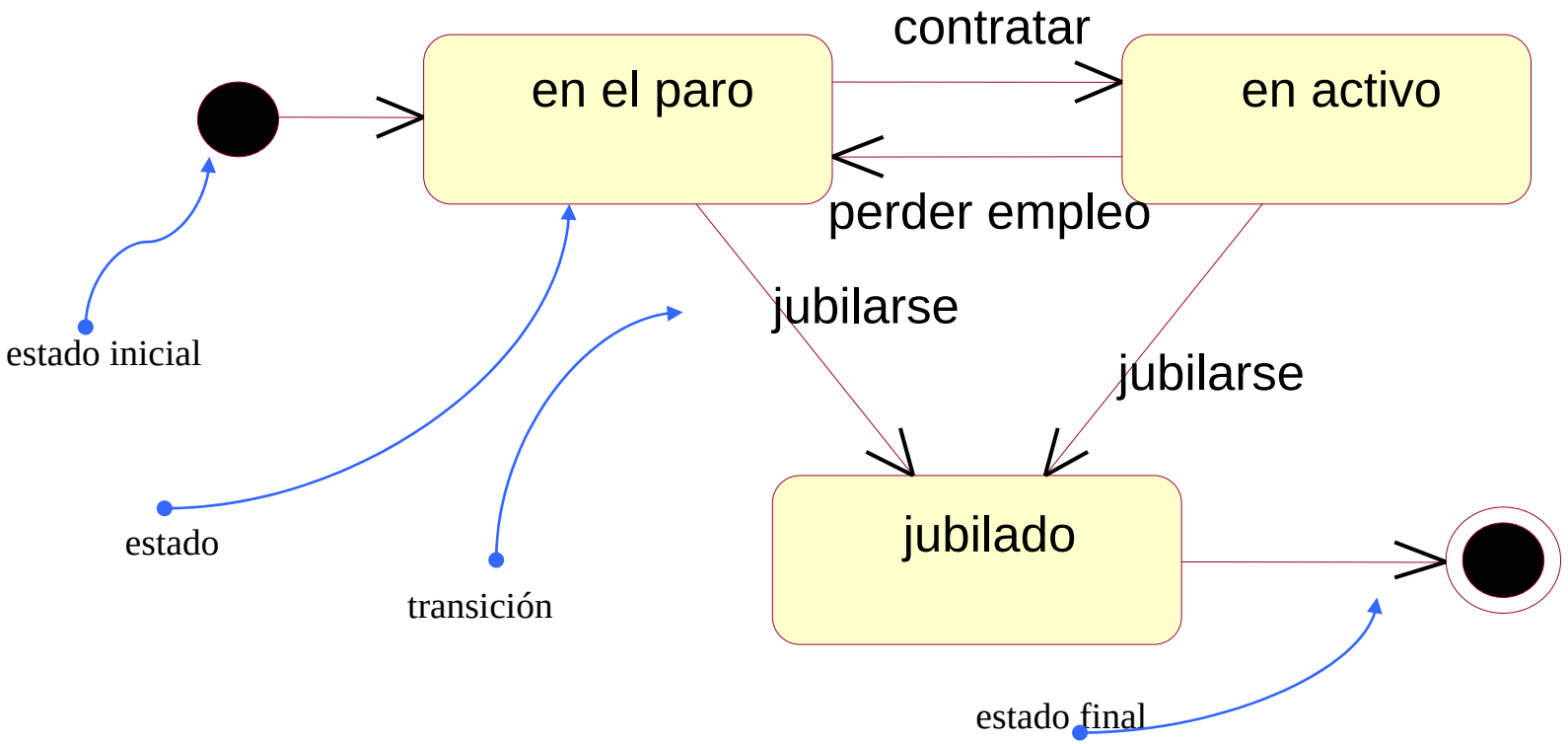
- Especifica la secuencia de estados por las que pasa un objeto a lo largo de su vida en respuesta a eventos, junto con sus respuestas a esos eventos.

- Útil si las instancias de una clase tienen un comportamiento que depende de su historia o que deben responder a eventos externos: objetos reactivos.

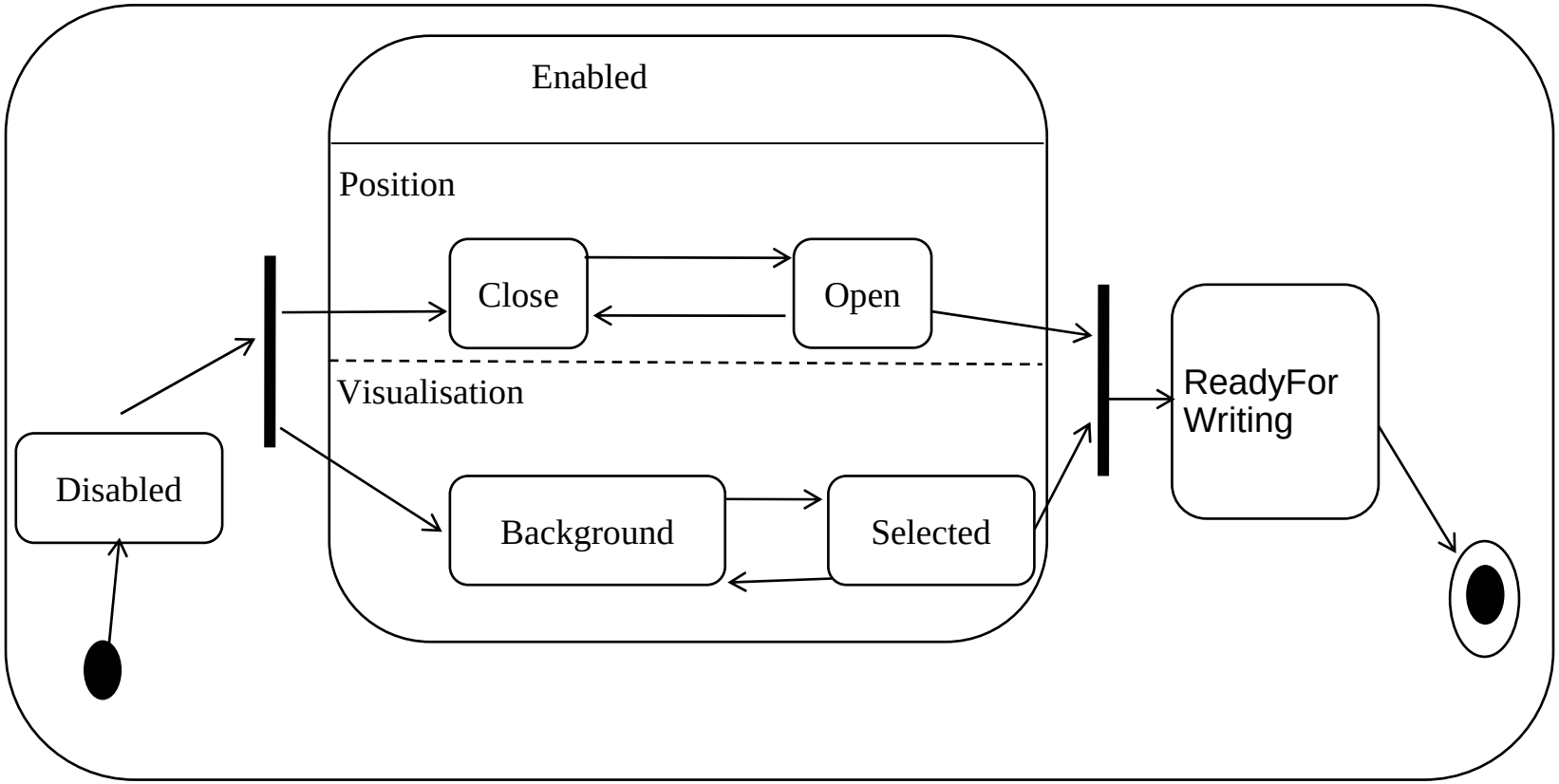
- Se representa mediante un diagrama de estados.

Ejemplo de Statechart

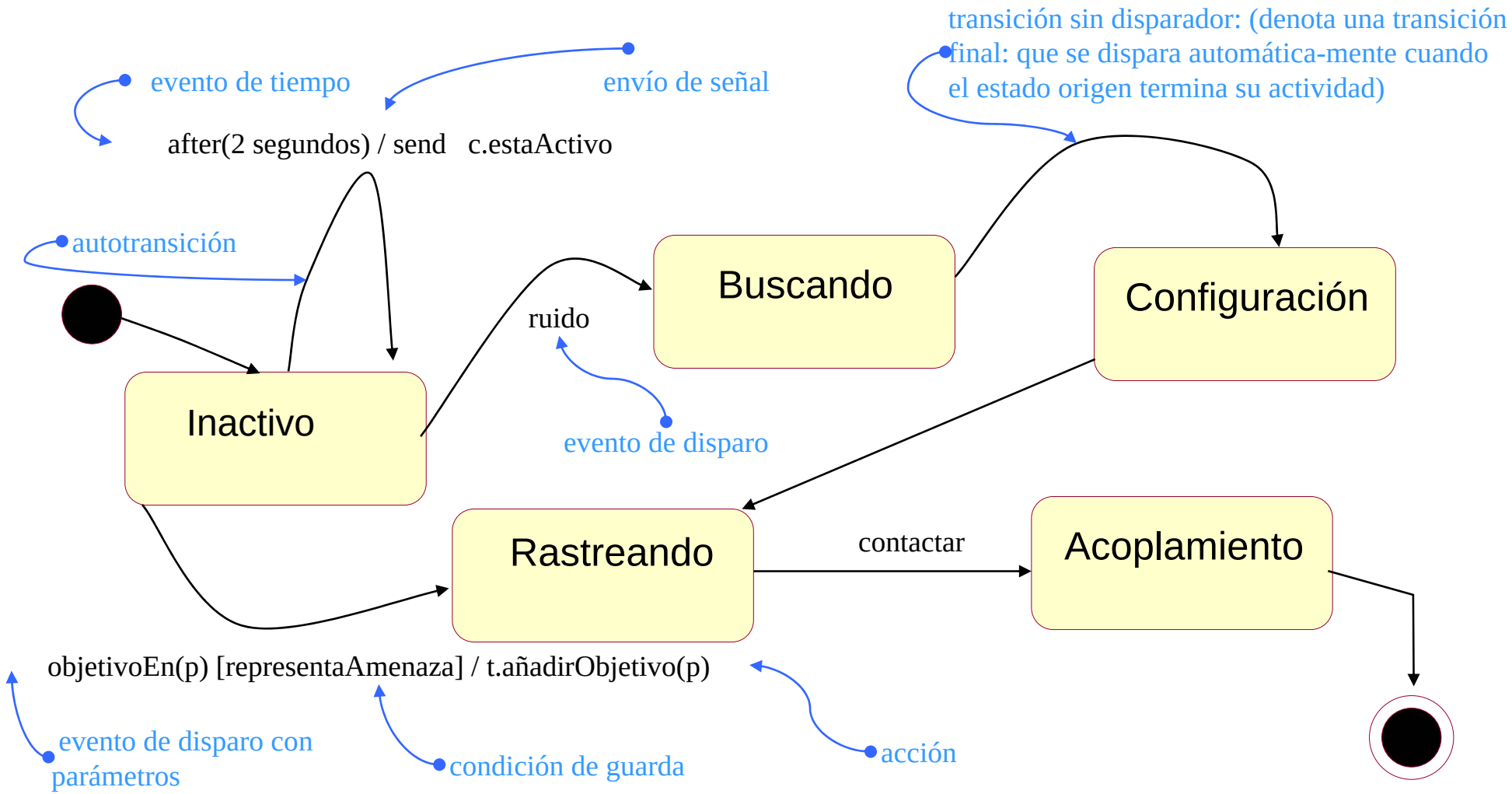
- Asociado a la clase Persona:

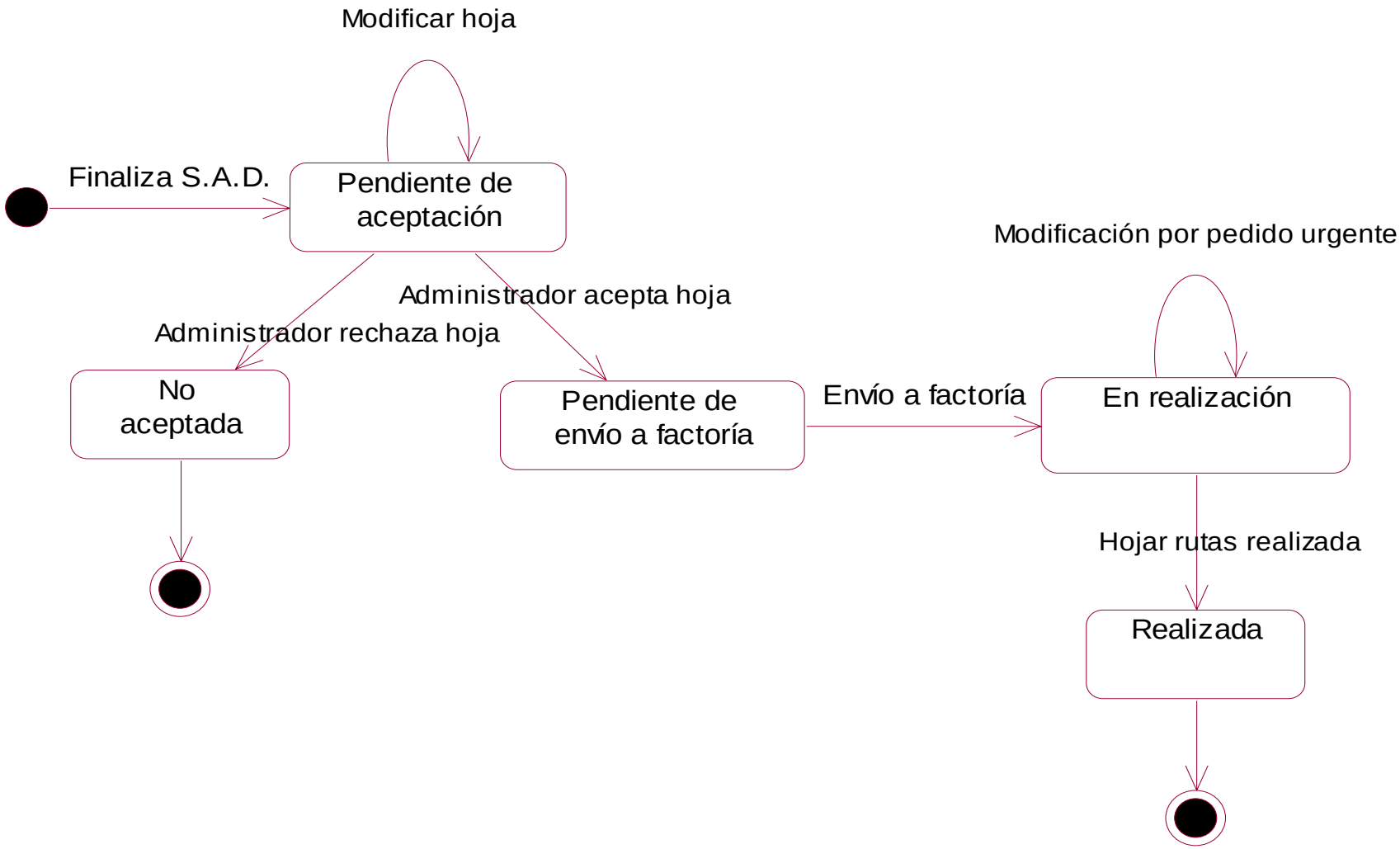


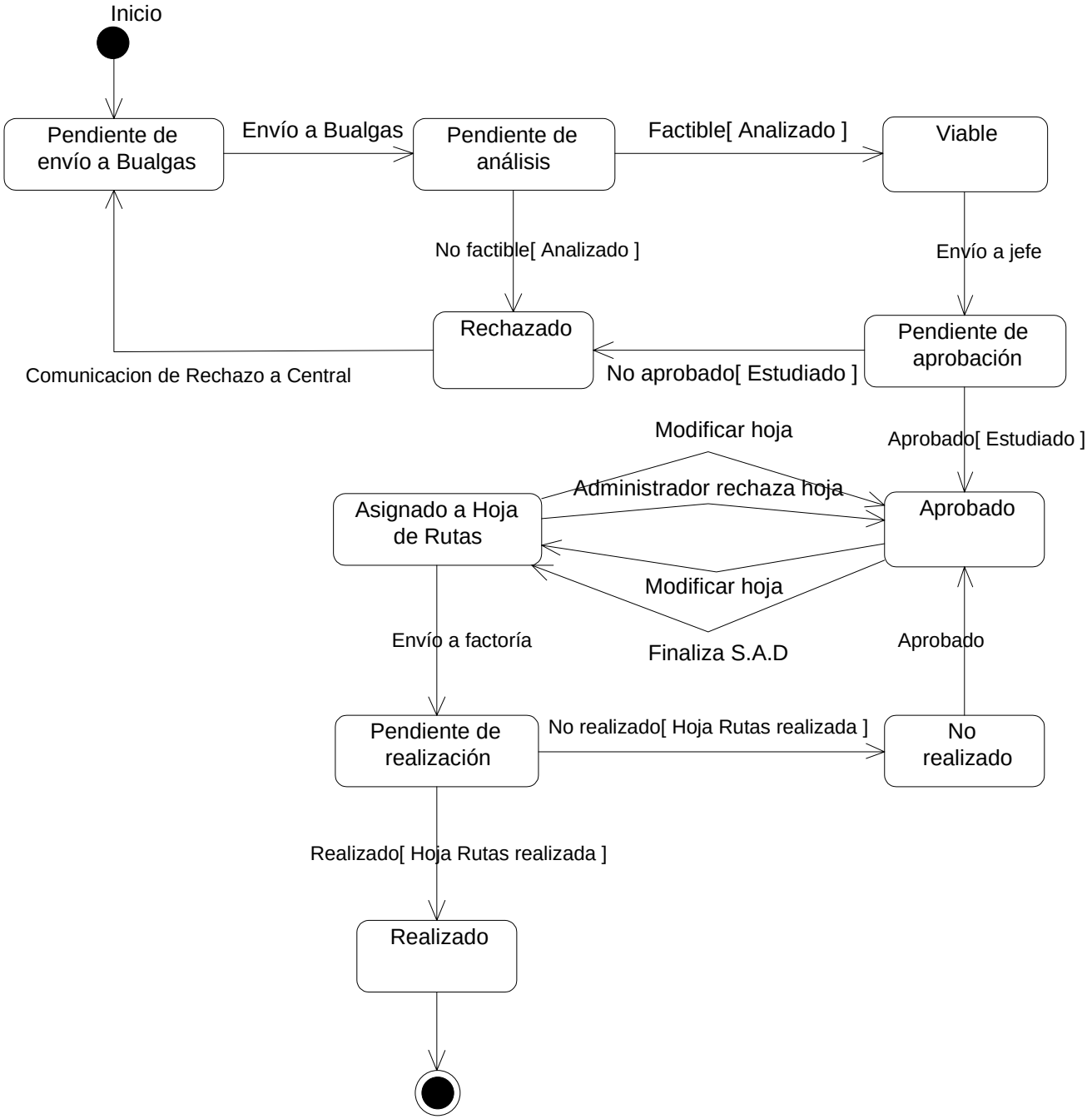
Ejemplo de Statechart



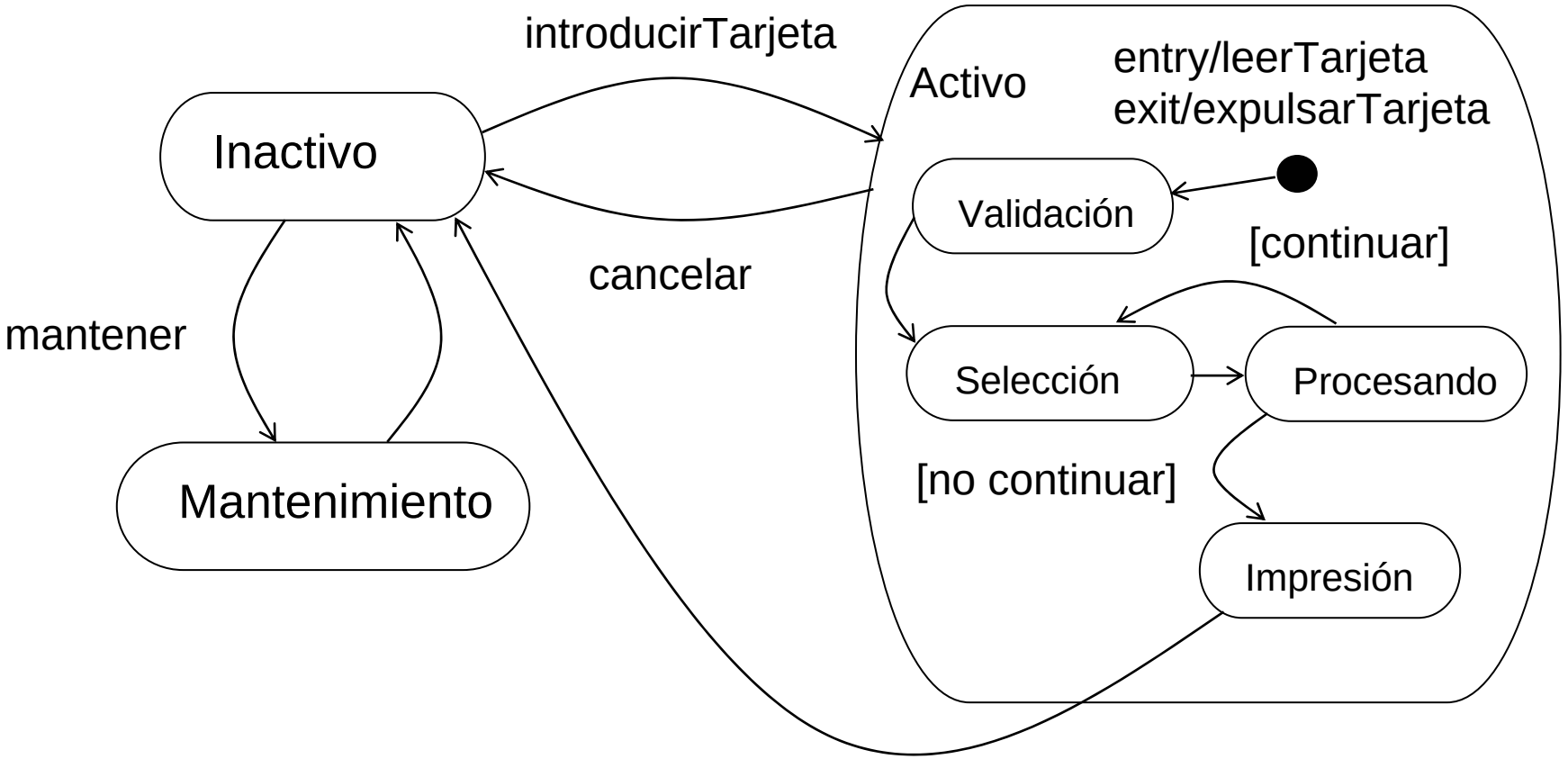
Transiciones simples y complejas


evento de tiempo
envío de señal
transición sin disparador: (denota una transición final: que se dispara automática-mente cuando el estado origen termina su actividad)
after(2 segundos) / send c.estaActivo
autotransición
Buscando
Configuración
ruido
Inactivo
evento de disparo
Rastreando
contactar
Acoplamiento
objetivoEn(p) [representaAmenaza] / t.añadirObjetivo(p)
evento de disparo con parámetros
condición de guarda
acción

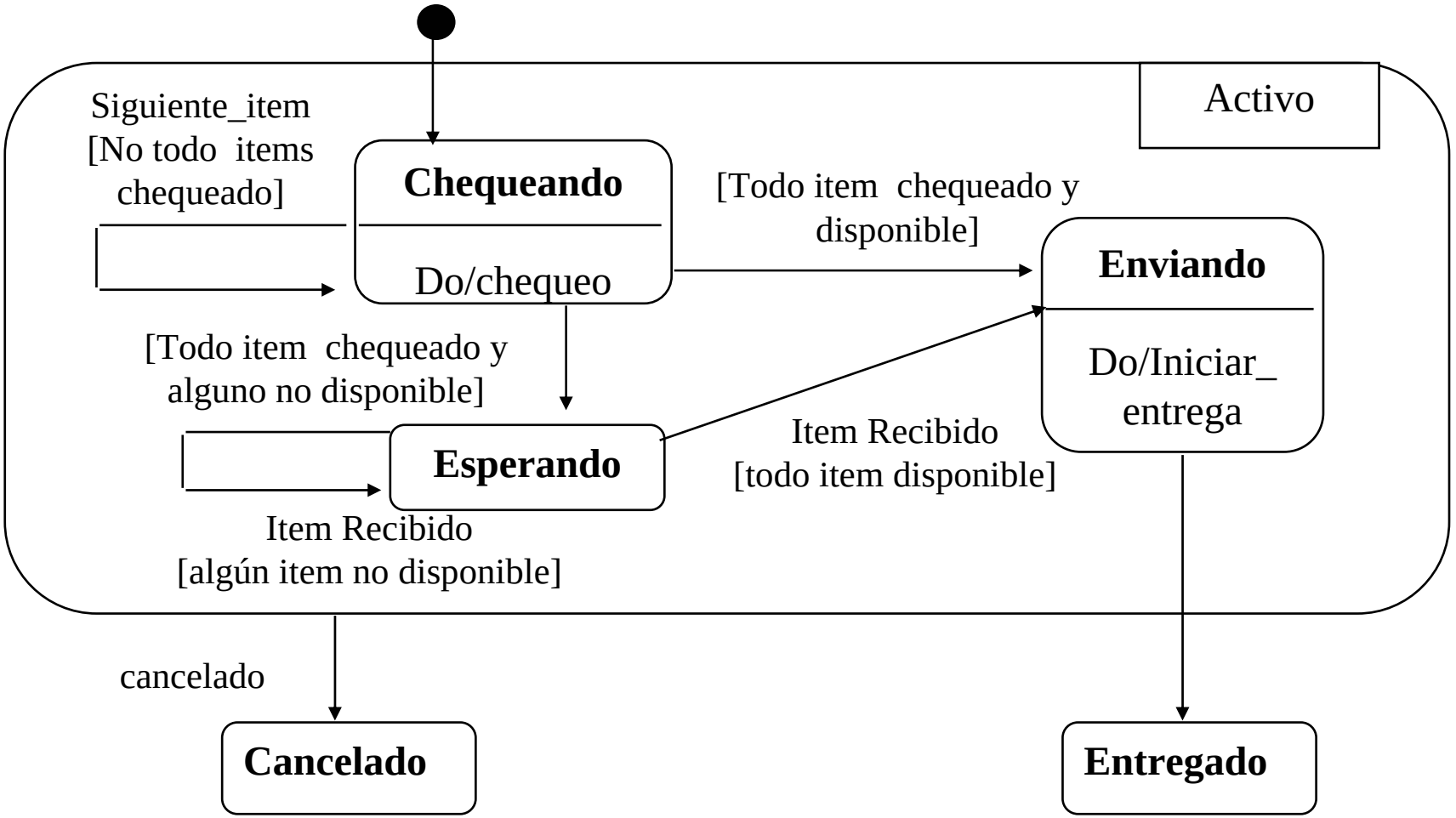

Modificar hoja
Finaliza S.A.D.
Pendiente de aceptación
Administrador rechaza hoja
Administrador acepta hoja
No aceptada
Pendiente de envío a factoría
Envío a factoría
Modificación por pedido urgente
En realización
Hojar rutas realizada
Realizada


Inicio
Pendiente de envío a Bualgas
Envío a Bualgas
Pendiente de análisis
Factible[Analizado]
Viable
No factible[Analizado]
Envío a jefe
Rechazado
Pendiente de aprobación
Comunicacion de Rechazo a Central
No aprobado[Estudiado]
Modificar hoja
Aprobado[Estudiado]
Administrador rechaza hoja
Asignado a Hoja de Rutas
Aprobado
Modificar hoja
Envío a factoría
Finaliza S.A.D
Aprobado
Pendiente de realización
No realizado[Hoja Rutas realizada]
No realizado
Realizado[Hoja Rutas realizada]
Realizado

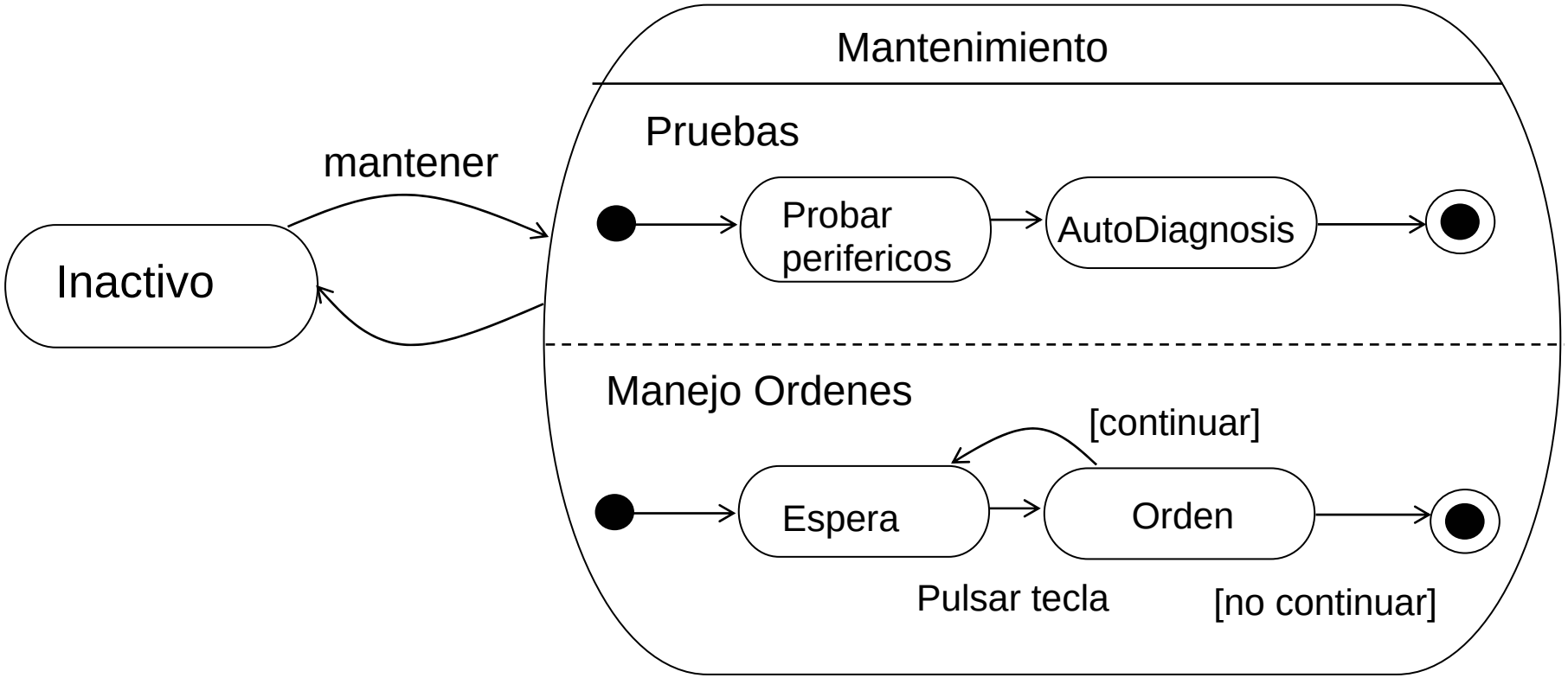
Subestados secuenciales



Subestados secuenciales



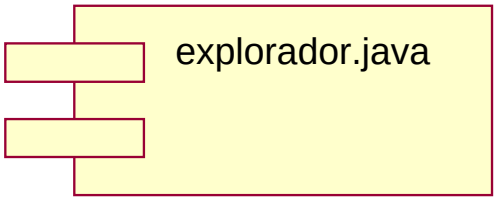
Subestados concurrentes

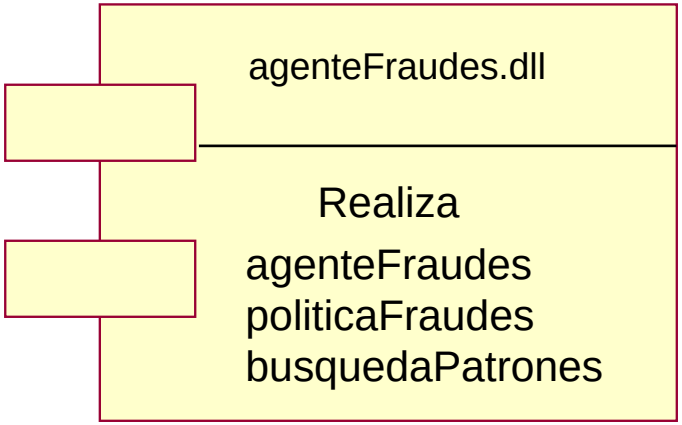


Componentes

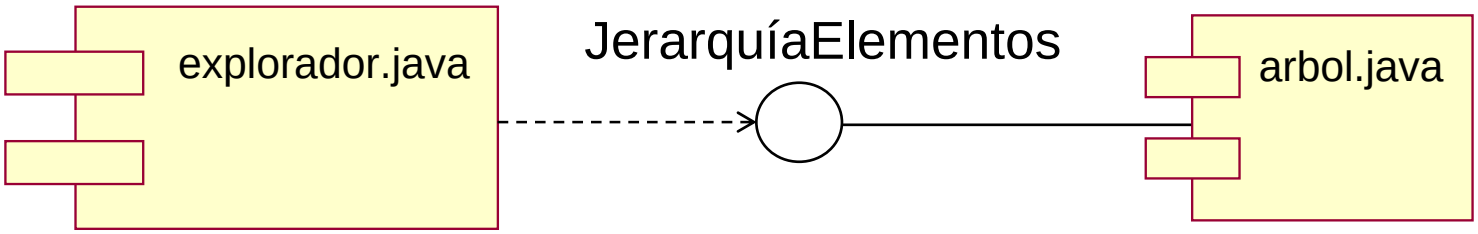
- Un componente es una parte **física** y **reemplazable** de un sistema, que conforma con un conjunto de interfaces y proporciona su implementación.
- Modela artefactos tales como: ejecutables, bibliotecas, tablas, ficheros, documentos,..
- Representa el empaquetamiento físico de elementos lógicos tales como clases, interfaces,..
- Residirán en los nodos del sistema

Componentes





Componentes



El componente *arbol.java* puede ser reemplazado por otro que proporcione la interfaz *JerarquíaElementos*.

Estereotipos: ***executable, library, file, table, document***

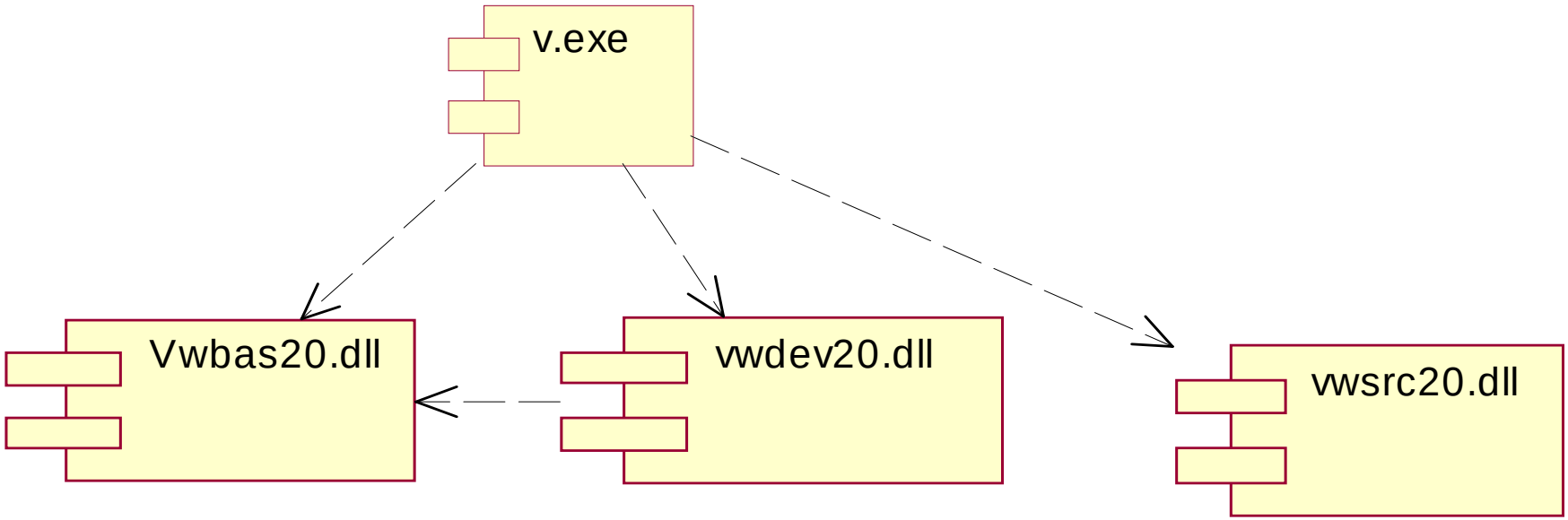
Tipos de componentes

- **Despliegue**
  - Necesarios y suficientes para formar un sistema ejecutable: librerías dinámicas(dll), ejecutables (exe),..
- **Productos del trabajo**
  - Permanecen al final del proceso de desarrollo: archivos código fuentes, ficheros de datos,..
  - Con ellos se crean los componentes de despliegue
- **De ejecución**
  - Se crean durante la ejecución: objeto COM, instanciado a partir de una dll.

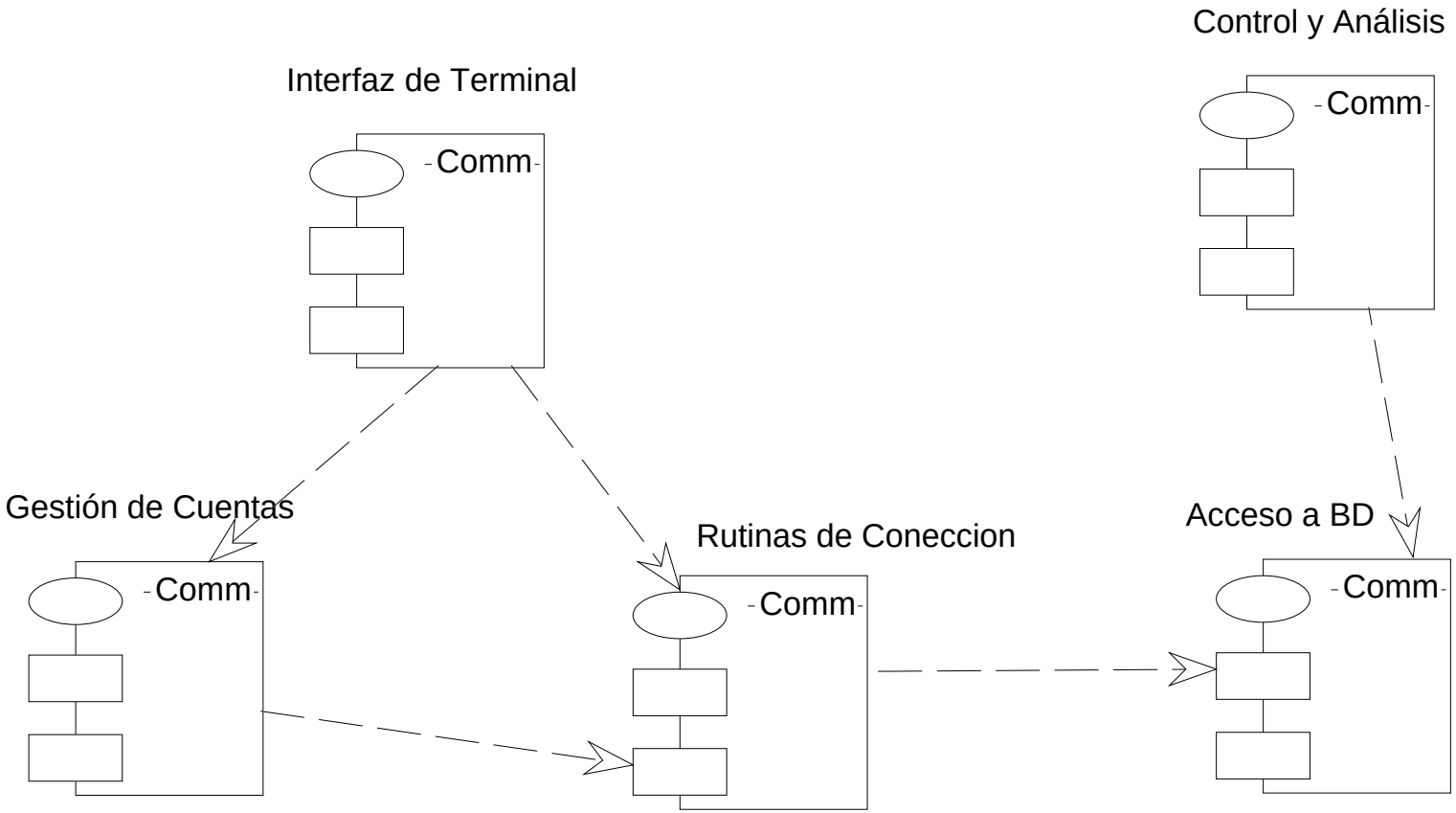
Diagrama de Componentes

- Modelado de ejecutables y bibliotecas
- Modelado de código fuente
- Modelado de una API

Modelado de ejecutables



Ejemplo


Control y Análisis
Comm
Interfaz de Terminal
Comm
Gestión de Cuentas
Comm
Rutinas de Coneccion
Comm
Acceso a BD
Comm

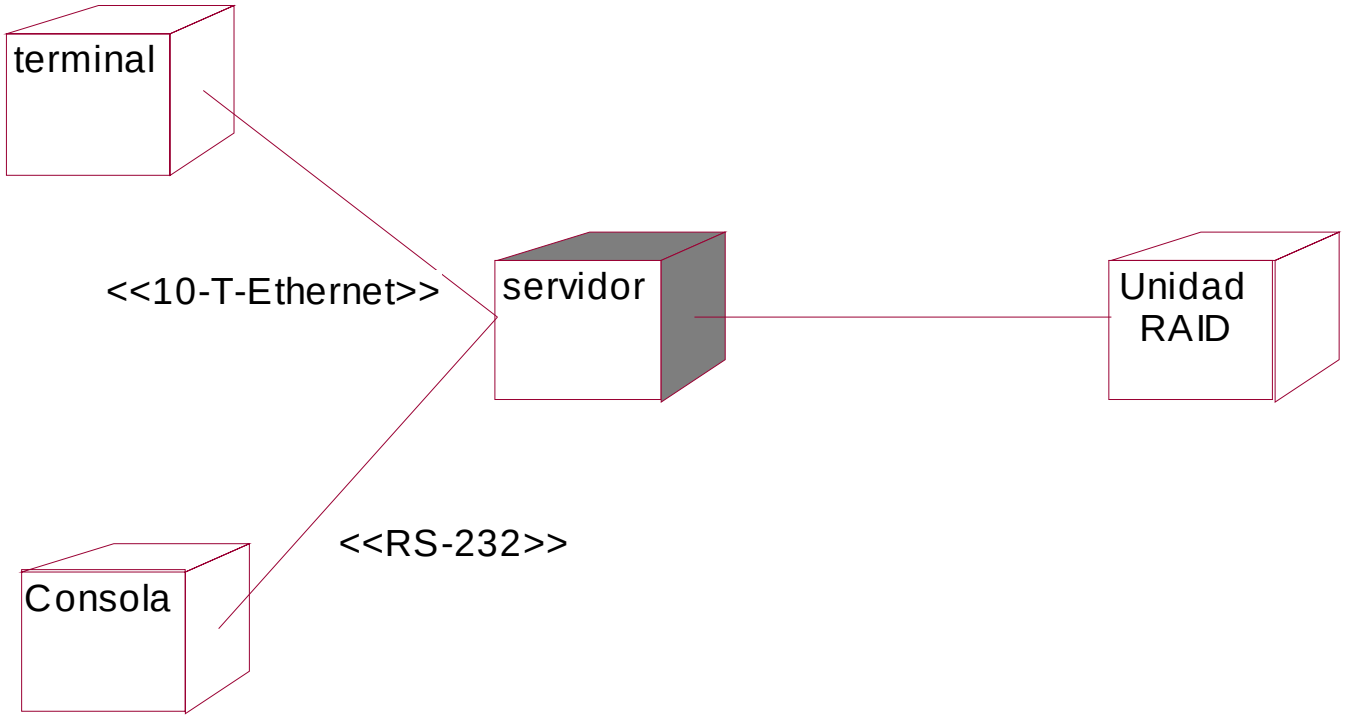
Nodos

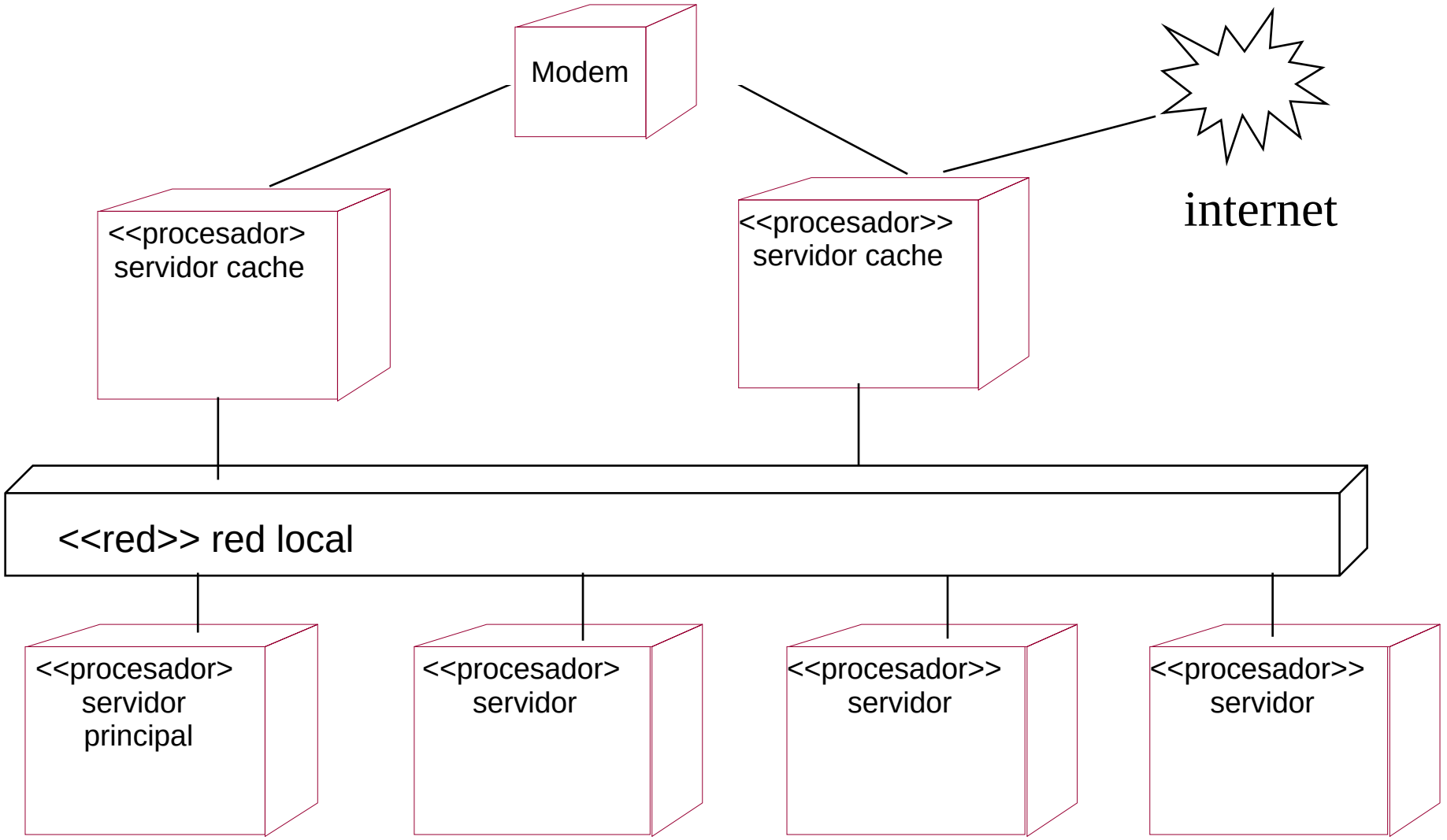
- Un nodo es un elemento físico que existe en tiempo de ejecución y representa un recurso computacional que puede tener memoria y capacidad de procesamiento.
- Los componentes se ejecutan en nodos
- Los nodos representan el despliegue físico de los componentes.

Diagramas de Despliegue

- Muestra la configuración de los nodos que participan en la ejecución y de los componentes que residen en los nodos.
- Incluye nodos y arcos que representan conexiones físicas entre nodos.
- Modelado de sistemas empotrados, sistemas cliente-servidor, sistemas distribuidos.

Diagrama de Despliegue




Modem
internet
<<procesador>
servidor cache
<<procesador>>
servidor cache
<<red>> red local
<<procesador>
servidor
principal
<<procesador>
servidor
<<procesador>>
servidor
<<procesador>>
servidor

Colaboraciones

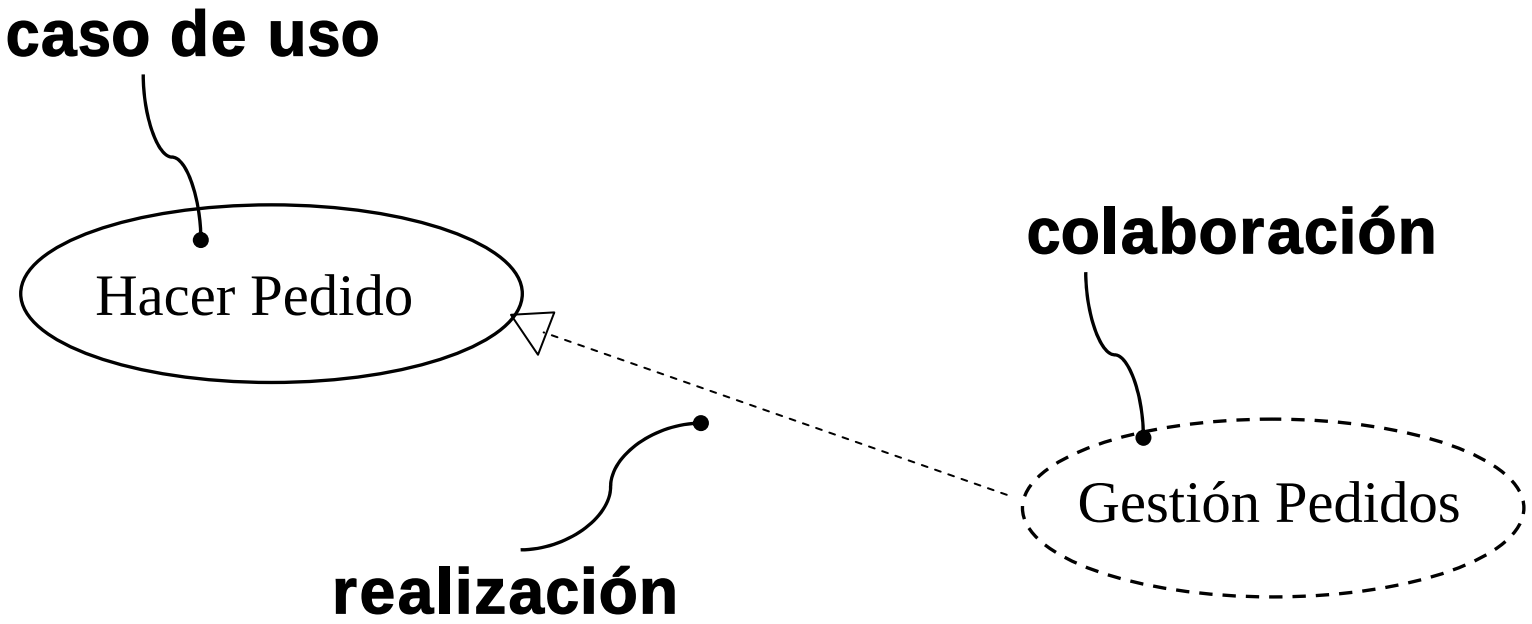
- Sociedad de clases, interfaces y otros elementos que colaboran para proporcionar un comportamiento cooperativo mayor que la suma de los comportamientos de los elementos.

- Parte **estructural** (diagrama de clases) y parte de **comportamiento** (diagrama de interacción).

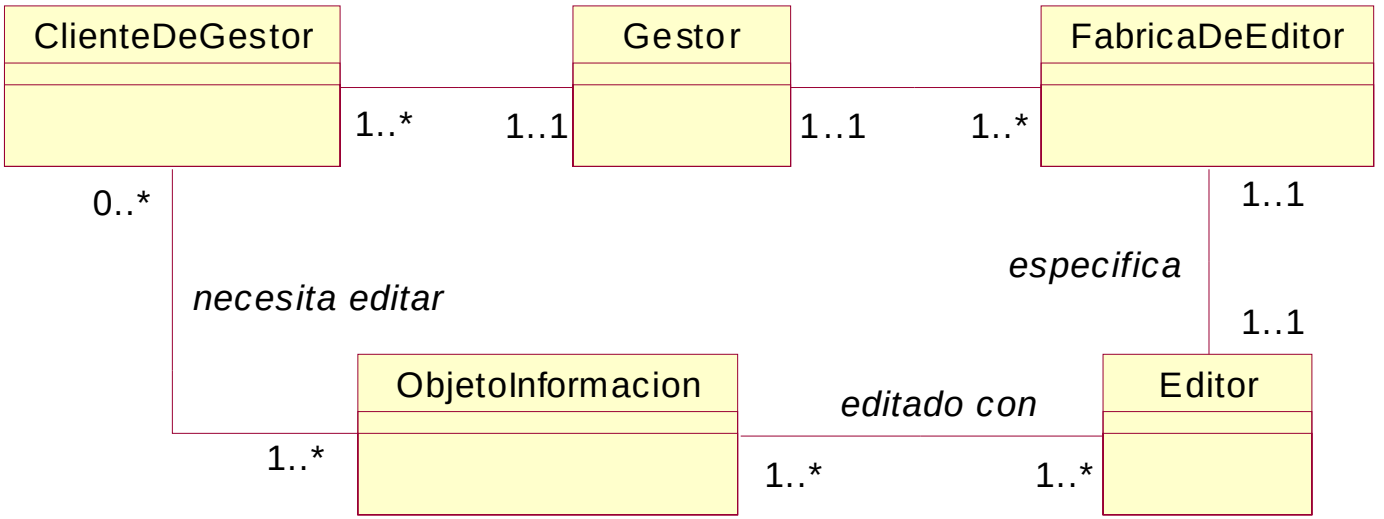
Colaboraciones

- El núcleo de la arquitectura de un sistema está formado por un conjunto de colaboraciones que representan las decisiones de diseño más importantes.
- Un sistema orientado a objetos bien estructurado se compone de un conjunto relativamente pequeño de colaboraciones.
- Modelado de un caso de uso, operación o mecanismo (patrón o *framework*)

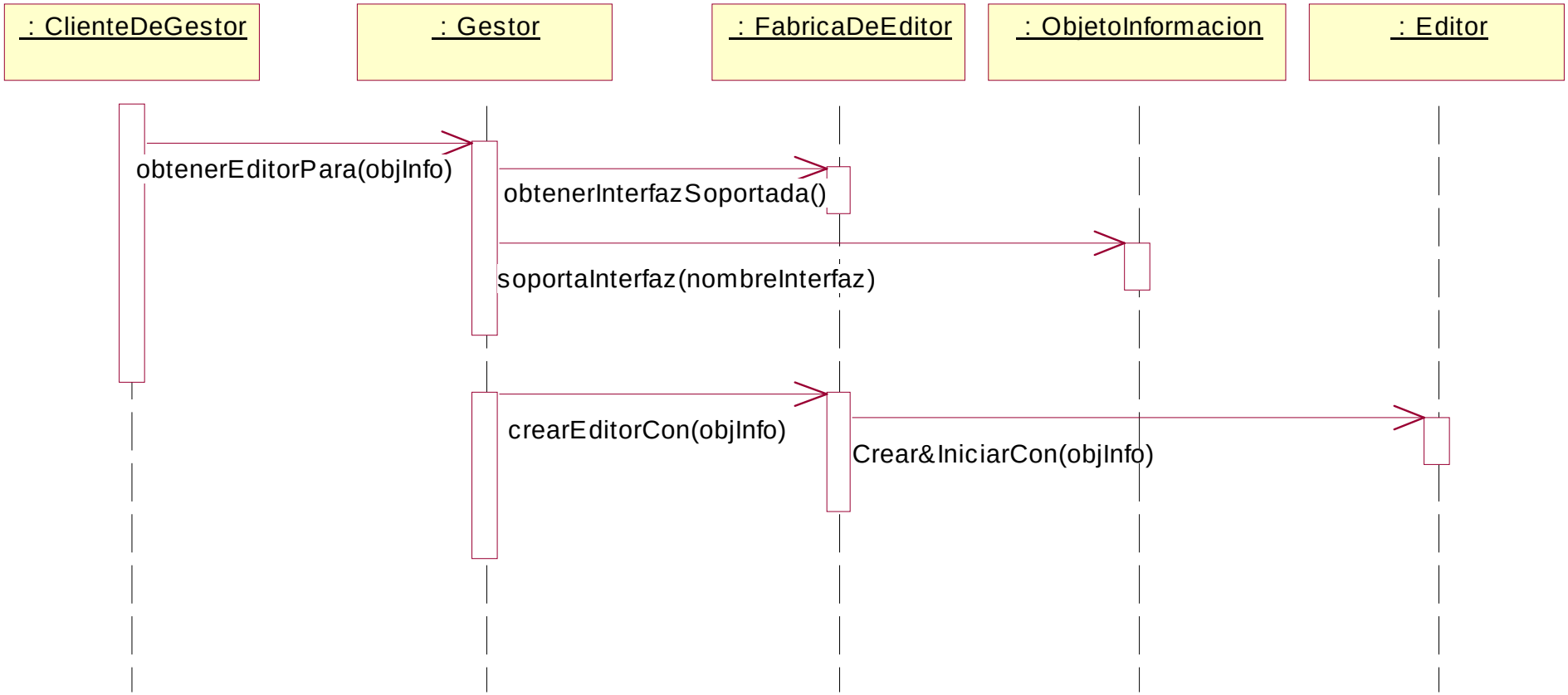
Casos de uso y Colaboraciones



Gestor Documentos (Parte estática)

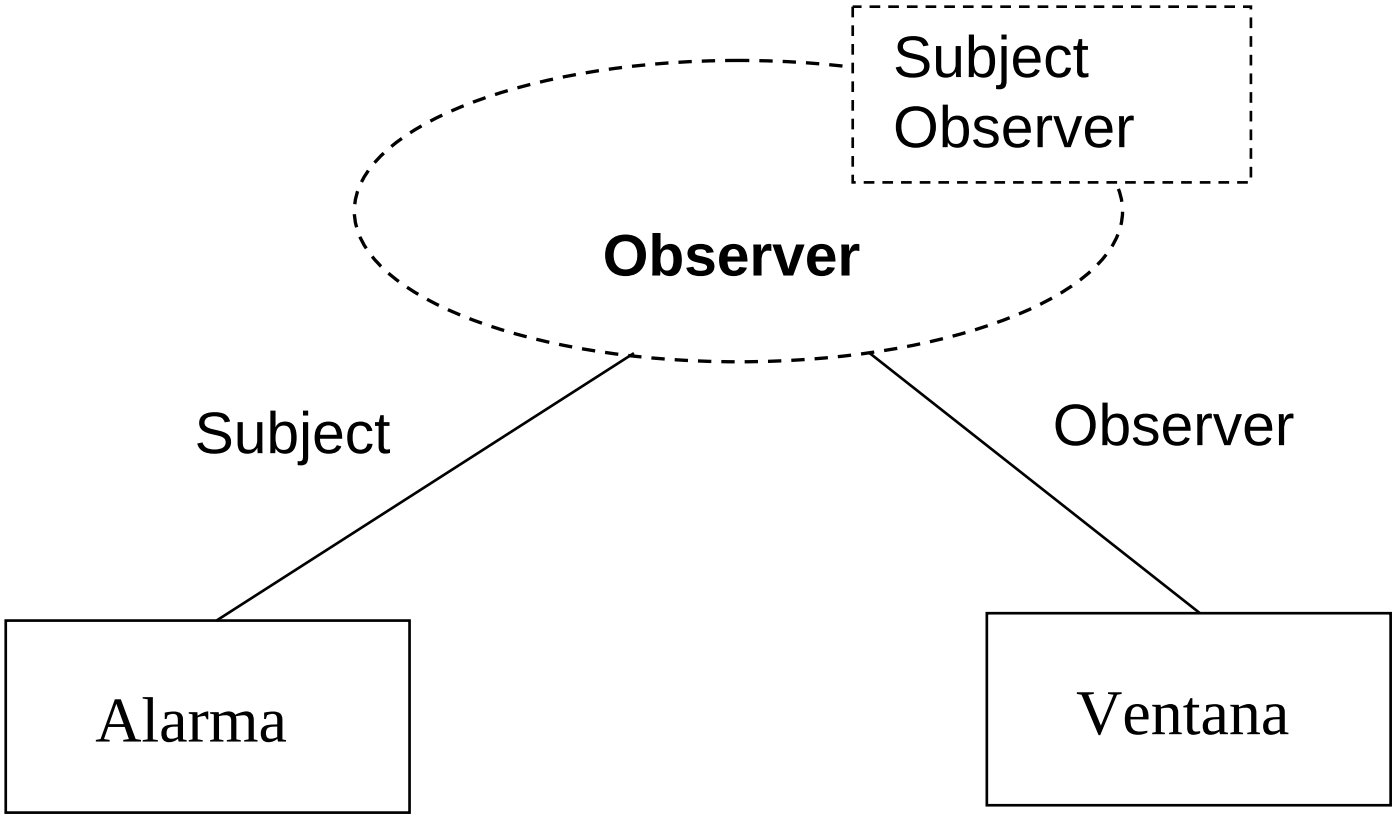


Gestor Documentos (Comportamiento)

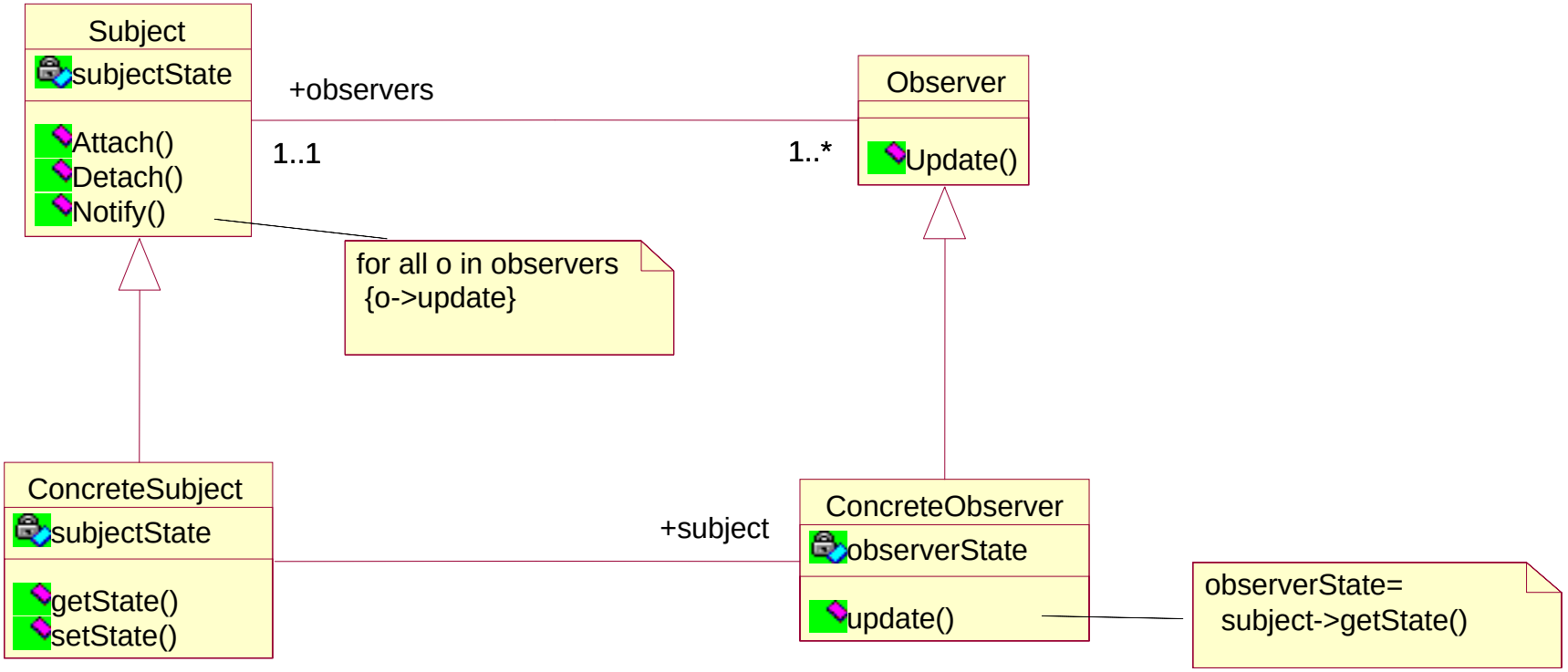


Colaboraciones Parametrizadas

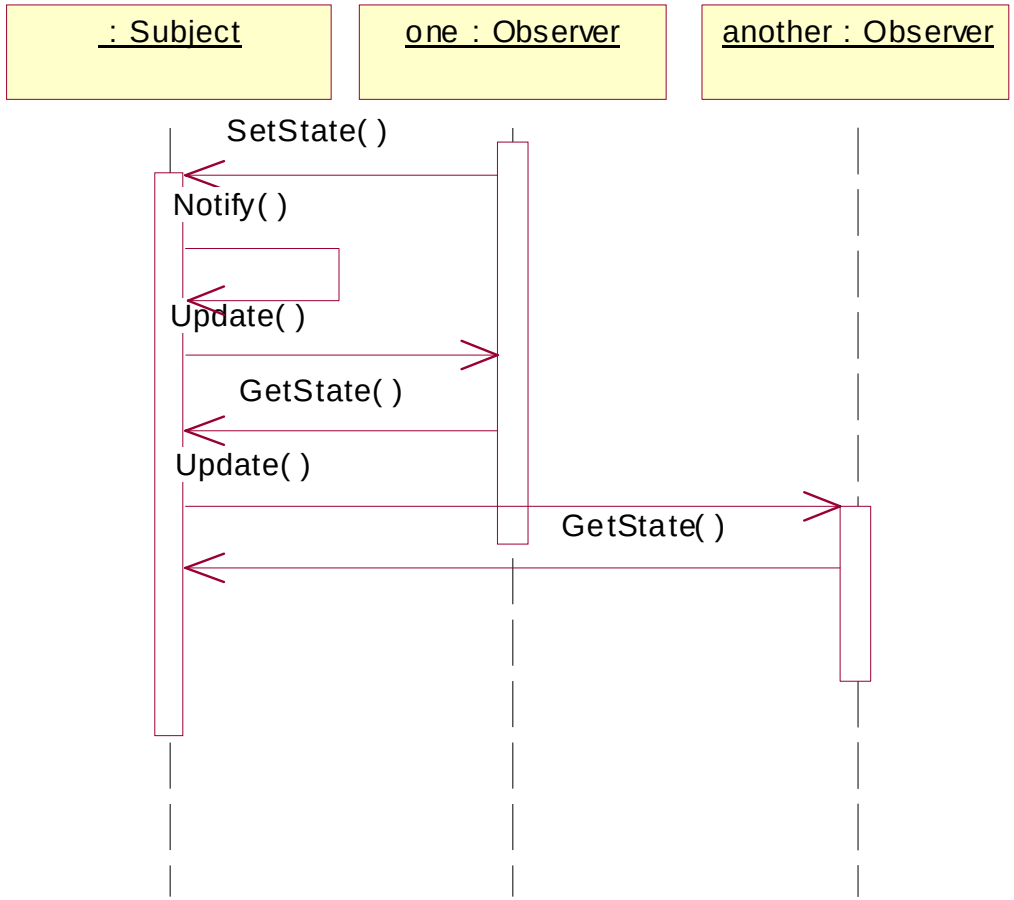
Modelado de patrones de diseño



Patrón de diseño (Parte Estática)



Patrón de diseño (Parte dinámica)



OCL (*Object Constraint Language*)

OCL puede ser usado por UML y otros lenguajes para **especificar** restricciones, expresiones de navegación, expresiones booleanas, consultas y otras expresiones incluidas en sus modelos.

- Lenguaje “formal” (más bien “semiformal”) para expresar restricciones libres de efectos colaterales.
- Las expresiones OCL no cambian el estado del sistema. No está destinado a escribir acciones o código ejecutable
- OCL es un lenguaje tipado.
- Desarrollado por Joe Warmer dentro de IBM.

Uso de OCL

- OCL puede ser usado con distintos fines:
  - Para especificar **invariantes** sobre clases y tipos en el modelo de clases.
  - Para especificar **pre** y **post-condiciones** sobre operaciones y métodos.
  - Para describir **guardas** (en statecharts y otros elementos).
  - Como lenguaje de navegación.
  - Para especificar restricciones sobre operaciones.
- OCL es utilizado también para especificar la semántica de UML.

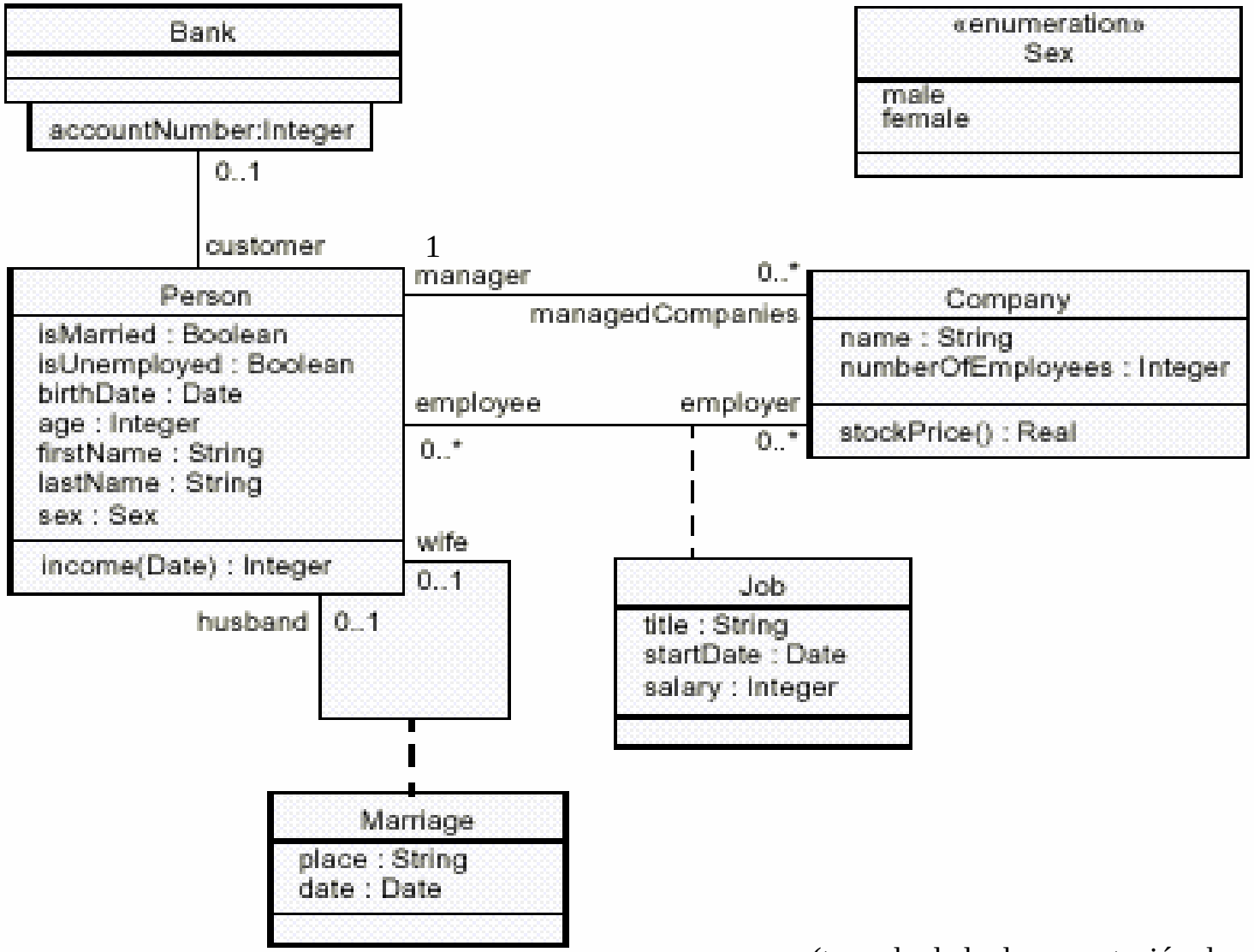


Figure 6-1 Class Diagram Example

(tomado de la documentación de UML v1.4)

OCL. *Self* y *context;* Invariantes

- Cada expresión OCL se escribe en el contexto de una instancia de un tipo específico. Se usa ***self*** para referirse a la instancia contextual. Por ejemplo, si el contexto es Company, ***self*** se refiere a una instancia de Company.
- Ejemplo:

```
context Company
 inv: self.numberOfEmployees > 50
```

Si el contexto es claro, se puede suprimir self. También se puede representar de forma alternativa con un nombre distinto como en:

```
context c : Company
 inv:c.numberOfEmployees > 50
```

OCL. Pre-, Post- condiciones de una operación

- Se usan las etiquetas ‘pre’ y ‘post’ antes de especificar las pre-y post-condiciones, que además pueden tener un nombre (*resultOK*). La declaración de operación, con sus parámetros, sigue a *context* y al tipo al que pertenece.
- *self* se referirá al objeto sobre el que se llama la operación
- ***result*** denota el resultado de la operación, si lo hay.
- Se pueden usar los nombres de los parámetros *(param1…)*.

```
context Typename::operationName(param1 : Type1, ... ): ReturnType
pre parameterOk: param1 > ...
post resultOk: result = ...
```

Tipos básicos y predefinidos

- Tipos básicos:
  - Boolean, Integer, Real y String.
  - Se complementan con:

    Collection, Set, Bag y Sequence así como los de

    Enumeration, OclExpression, OclType, OclState y OclAny

Tipos predefinidos

- Tipos Básicos: Existen una serie de tipos básicos predefinidos (y operaciones sobre ellos) en OCL, que son: Boolean, Integer, Real, String.

type	operations
Integer	*, +, -, /, abs
Real	*, +, -, /, floor
Boolean	and, or, xor, not, implies, if-then-else
String	toUpper, concat

- *Collection*, *Set*, *Bag* y *Sequence* se consideran también tipos básicos

Type	Conforms to/Is a subtype of
Set(T)	Collection(T)
Sequence(T)	Collection(T)
Bag(T)	Collection(T)
Integer	Real

Propiedades: Atributos

Ejemplo: la edad de una Persona se escribe como *self.age:*

```
context Person inv:
self.age > 0
```

El valor de la subexpresión *self.age* es el valor del atributo *age* sobre el objeto *self*. El tipo de esta subexpresión es el tipo del atributo *age* (en este caso tipo básico *Integer*)

El invariante de arriba está expresando que

"la edad de una persona es siempre mayor que cero"

Propiedades: Operaciones

Ejemplo: un objeto persona tiene una operación *income* que es función de fecha (*Date*).

Para un persona *aPerson* y una fecha *aDate*, esta operación sería accedida como:

aPerson.income(aDate)

La operación puede ser definida con una postcondición. El objeto devuelto por ésta (el resultado) se denomina *result:*

context Person::income (d: Date) : Integer

post: result = age * 1000

Se ponen paréntesis aunque la operación o método no tenga argumentos:

context Company **inv**:

self.stockPrice() > 0

Propiedades: Extremos de asociación y navegación

Ejemplo: context Company

inv: self.manager.isUnemployed = false

inv: self.employee->notEmpty

En el primer invariante, *self.manager* es una persona (la multiplicidad de la asociación es 1)

En el segundo, *self.employee* se evaluará a un conjunto de personas. La expresión completa es un valor booleano que vale true cuando la colección a la izquierda es no vacía.

Si la asociación en el Diagrama de Clases está adornada con **{ordered},** en lugar de un **conjunto** (**valor por defecto** de la navegación) obtendríamos una **secuencia** como resultado

Propiedades: Extremos de asociación y navegabilidad

- Se puede acceder a las propiedades de los tipos colección (Sets, Bags, Sequences) mediante una flecha '->' seguida por el nombre de la propiedad:

```
context Person
 inv: self.employer->size < 3
```

Aplica la propiedad *size* sobre el conjunto (Set) *self.employer*, que resulta en el número de "empleadores" de la persona *self*.

```
context Person
 inv: self.employer->isEmpty
```

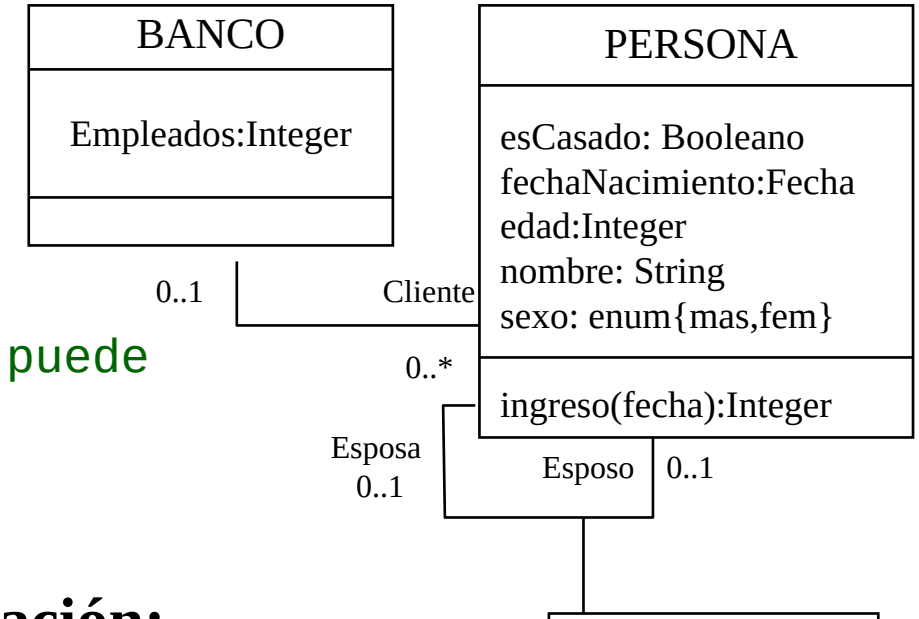
Aplica la propiedad *isEmpty* sobre el conjunto *self.employer*. Se evalúa a true si el conjunto de empleadores es vacío o false en caso contrario.

OCL. Más ejemplos

Atributos:

```
context Persona inv:
self.edad>0
context Banco inv límEmpleados:
self.empleados < 100
```

Opcionalmente, la restricción puede tener un nombre, en este caso "límEmpleados"



Extremos de asociación y Navegación:

```
context Persona inv:
self.esposa->notEmpty implies
self.esposa.sexo=femenino
```

implies es una operación definida sobre el tipo de datos predefinido de OCL, Boolean

OCL. Más ejemplos

Operaciones:

```
context
 Persona::ingreso(fecha):Integer
post: result > 1000
```

```
context Person::birthdayHappens()
post: age = age@pre + 1
```

El valor de una propiedad en una postcondición es el valor después de haberse completado la operación. Para referirse al valor de la propiedad al principio de la operación, se usa el postfijo '*@pre*', detrás del nombre de la propiedad.

OCL. Colecciones

- Operaciones de las Colecciones.
  - Seleccionar y Descartar (Select, Reject)

```
context Banco inv:
self.cliente->select(edad>50) --cto de personas
                              --mayores de 50
```

  - Recolectar (Collect)

```
context Banco inv:
self.cliente->collect(fechaNacimiento) -- bolsa con
          --las fechas de nacimiento de los empleados
```

OCL. Colecciones

- ParaTodos (ForAll)

```
context Banco inv:
self.cliente->forAll(Nombre='Juan') -- verdadero si
                    --todos los clientes se llaman Juan
```

- Existe (Exists)

```
context Company inv:
self.employee->exists( forename = 'Jack' )
context Company inv:
self.employee->exists( p | p.forename = 'Jack' )
context Company inv:
self.employee->exists( p : Person | p.forename = 'Jack' )
```

Estas expresiones se evaluan a *true* si la característica *forename* de al menos un empleado es igual a 'Jack.'